U.PORTO

MESTRADO
MULTIMÉDIA - ESPECIALIZAÇÃO EM CULTURA E ARTES

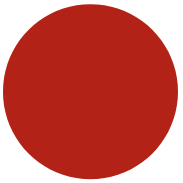
Construção de modelos neurais para a criação de arte generativa visual

Daniel Leal Machado

M
2018

FACULDADES PARTICIPANTES:

FACULDADE DE ENGENHARIA
FACULDADE DE BELAS ARTES
FACULDADE DE CIÊNCIAS
FACULDADE DE ECONOMIA
FACULDADE DE LETRAS



Construção de modelos neurais para a criação de arte generativa visual

Daniel Leal Machado

Mestrado em Multimédia

Orientador: Miguel Carvalhais (Prof. Dr.)

Co-orientador: Rui Rodrigues (Prof. Dr.)

Junho de 2018

Construção de modelos neurais para a criação de arte generativa visual

Daniel Leal Machado

Mestrado em Multimédia

Aprovado em provas públicas pelo Júri:

Presidente: Bruno Giesteira
Arguente: Eduardo Pereira
Vogal: Miguel Carvalhais

Junho de 2018

Abstract

The emergence of the generative adversarial networks (GANs) in 2014, allowed the development of neural models that effectively solve problems of computer vision such as: the generation of photorealistic images, the translation from image to image and the increasing of their resolution. Since then, several computational artists have been creatively exploring these generative models and presenting very different results, mainly in the visual field. However, the potential of the application of these models in generative art has been hampered by the inability to generate vector images and by the low resolution and predictability of the results. In this work are presented two neural models, called AGAN and bezierGAN, that attenuate these problems. The AGAN is an adversarial neural model with two discriminators that allows the generation of images with different distributions from the training image ones, thus reducing the degree of predictability of the results. The results obtained show that the images generated by the model are abstractions of the training images and that their intensity is controllable. The bezierGAN model is an adversarial neural model that generates vectorized images consisting of bézier curves and trained with bitmap images. The results obtained show that the model generates sketches with main lines that are defined by the distribution of the training images. The model has the potential to create a latent space for vectorized images, allowing the creation of minimalist videos and images with organic distributions in any resolution.

Keywords: Visual generative art , generative adversarial networks, AGAN, bezierGAN

Resumo

O aparecimento das redes generativas adversariais (GANs), no ano de 2014, permitiu o desenvolvimento de modelos neurais que solucionam eficazmente problemas da visão computacional como: a geração de imagens fotorealistas, a tradução de imagem para imagem e o aumento da resolução destas. Desde então, vários artistas computacionais têm vindo a explorar criativamente estes modelos generativos e a apresentar resultados muito diversos, principalmente na vertente visual. No entanto, o potencial de aplicação destes modelos na arte generativa tem sido travado pela incapacidade de geração de imagens vetoriais e pela baixa resolução e previsibilidade dos resultados. Neste trabalho são apresentados dois modelos neurais, designados por AGAN e bezierGAN, que contrariam esses problemas. O AGAN é um modelo neural adversarial com dois discriminadores que permite gerar imagens com distribuições diferentes das distribuições das imagens de treino, reduzindo, assim, o grau de previsibilidade dos resultados. Os resultados obtidos demonstram que as imagens geradas pelo modelo são abstrações das imagens de treino e que a intensidade destas é controlável. O modelo bezierGAN é um modelo neural adversarial treinado com imagens bitmap e que gera imagens vetorizadas constituídas por curvas de bézier. Os resultados obtidos demonstram que o modelo gera *sketches* com linhas principais definidas pela distribuição das imagens de treino. O modelo tem como potencialidade a criação de um espaço latente para imagens vetorizadas, permitindo a criação de vídeos e imagens minimalistas com distribuições orgânicas e em qualquer resolução.

Keywords: Arte generativa visual, redes neurais adversariais, AGAN, bezierGAN

Conteúdo

Lista de Figuras

Introdução

Contexto e motivação

O sucesso dos modelos baseados em redes neurais na resolução dos mais variados problemas da visão computacional [37, 15, 44, 24, 32] está a desencadear, na comunidade dos artistas computacionais, um interesse crescente no uso destes sistemas para a geração de obras de arte. A partir do ano de 2014, no qual surgiram os primeiros modelos neurais com propósito artístico, como o *DeepDream*[1], os artistas computacionais começaram a acompanhar os progressos científicos desta área e a explorar artisticamente os códigos disponibilizados pelas equipas de investigadores. O grande número de modelos propostos, assim como as suas respetivas aplicações, como a geração de imagem [40] e áudio [52], transferência de estilo [24], super-resolução [31] e previsão do próximo *frame* [32], permitem que os artistas obtenham uma grande diversidade de resultados. Para além disso, estes modelos são baseados em algoritmos de aprendizagem profunda[2] e, portanto, aprendem a realizar a tarefa para a qual foram desenhados através de exemplos de treino. Deste modo, os artistas podem treinar os modelos a produzir obras com os objetos e estilos que selecionam através dos exemplos de treino. Por estas razões, os modelos neurais são sistemas generativos que dão possibilidades únicas dentro do contexto da arte generativa. Torna-se, assim, fundamental estudar o modo como os artistas computacionais têm usado os modelos neurais nos últimos anos, identificar os problemas que estes encontram ao aplicá-los e construir modelos experimentais, baseados nos modelos científicos, de modo a aumentar as possibilidades artísticas destes novos sistemas generativos.

Objetivos

O presente trabalho tem como objetivo a construção de dois modelos neurais que combatem os problemas que surgem quando este tipo de modelos são aplicados no contexto da arte generativa. Os problemas em causa são: a previsibilidade dos resultados, incapacidade de composição por

[1] O *DeepDream* é, originalmente, um programa que amplifica as caraterísticas que uma rede neural deteta numa dada imagem. Este processo de amplificação dá origem a uma imagem sobre-processada com uma estética tendencialmente psicadélica. O programa foi criado pelo engenheiro da *Google*, Alexander Mordvintsev.

[2] Sub-ramo da aprendizagem máquina que consiste em algoritmos capazes de aprender a construir vários niveís de representação dos dados, de modo a modelar relações complexas entre eles [7].

elementos (p.ex. retângulos, linhas, círculos, etc) e baixa resolução. Estes problemas são identificados após uma análise teórica dos diferentes usos dados aos modelos neurais, no contexto da arte generativa visual (sec.1.3). Esta análise teórica é um objectivo secundário do trabalho, na medida que não é apenas uma revisão de literatura, mas também uma tentativa de fazer uma cronologia do aparecimento dos diferentes modelos neurais na arte generativa visual e identificar o modo como os artistas os usam para a geração de imagens. No final desta análise, surgem as questões de investigação, derivadas dos problemas já referidos, que constituem os problemas centrais desta dissertação:

1. É possível criar um modelo neural que gere imagens que consistem em abstrações das imagens de treino, de modo a combater a previsibilidade dos resultados dos modelos neurais generativos usados actualmente?

2. É possível criar modelos neurais para geração de imagem baseada em elementos vetoriais e que possam ser treinados com imagens *bitmap*[3]?

Metodologia

De modo a responder às questões 1 e 2 anteriormente apresentadas, são propostos dois modelos neurais generativos designados AGAN e bezierGAN. O modelo AGAN foi desenhado para responder à questão 1 e consiste num modelo neural que captura uma distribuição de dados[4], determinada pelas distribuições dos dados de treino, mas que não é igual ou assintoticamente próxima a estas. Isto permite que o modelo gere imagens que preservam algumas caraterísticas das imagens de treino, mas que contenham simultâneamente outras que não estejam presentes. A ideia central é que o modelo tente "inspirar-se" nas imagens de treino em vez de tentar copiá-las, construindo abstrações destas.

O modelo bezierGAN foi desenhado para responder à questão 2 e consiste num modelo neural que gera imagens vetoriais baseadas em curvas de bézier. Este modelo gera imagens com curvas de bézier dispostas no espaço dos píxeis, de modo a capturar a distribuição das imagens *bitmap* de treino.

Cada um dos modelos é avaliado para dois experimentos, nos quais é treinado com diferentes bibliotecas de imagens. Os resultados dos dois experimentos são, então, avaliados qualitativamente e as conclusões são feitas relativamente ao desempenho de cada um dos modelos.

[3]Mapeamento de uma grelha retangular de píxeis para um domínio de *bits* que representa os valores de cada um dos canais de cor, como por exemplo os canais RGB ou RGBA.

[4]Um rede neural generativa é uma função que mapeia um conjunto de vetores iniciais para um conjunto de vetores finais que têm uma distribuição de dados assintoticamente próxima da distribuição dos dados de treino. Se os dados de treino forem imagens, esta distribuição representa as probabilidades de cada píxel i ter uma dada cor j.

Resultados esperados

Para o modelo AGAN, esperam-se resultados que demonstrem que o modelo está a gerar imagens que sejam abstrações das imagens de treino, i.e., que preservem a estrutura básica destas, mas que haja espaço para variações de estilo e substituições de elementos. Entre as imagens geradas e as imagens de treino deve haver um grau de analogia, em vez de um grau de identidade, ao contrário do que se observa nos resultados obtidos com os modelos generativos usados actualmente, cujo objetivo é obter novas imagens que sejam estrutural e esteticamente idênticas às imagens de treino.

Relativamente ao modelo bezierGAN, esperam-se resultados que demonstrem que o modelo está a localizar as curvas de bézier no espaço dos píxeis, de modo a capturar a distribuição das imagens *bitmap* de treino. Como o modelo, à partida, está limitado computacionalmente a um número reduzido de curvas de bézier, espera-se que o modelo capture mais eficazmente as distribuições de imagens mais simples, i.e. imagens com elementos de área reduzida e com poucos pormenores, preferencialmente baseados em linhas. Um dos resultados mais importantes que se espera obter deste modelo é que o mapeamento entre os vetores de entrada e as imagens vetoriais de saída seja suave[5], de modo a permitir aos artistas a construção de vídeos coerentes, através da animação suave das curvas de bézier no espaço. Por último, pretende-se que este modelo seja generalizável, i.e. que desenvolva uma *framework* que permita adaptar o modelo para a geração de imagens baseadas noutros tipos de elementos.

[5] Funções suaves são funções que possuem derivadas de todas as ordens.

Capítulo 1

Revisão de literatura

1.1 Redes neurais: um sistema emergente na arte generativa computacional

Nos últimos anos, as redes neurais têm recebido uma grande atenção por parte das diferentes comunidades científicas, havendo um constante desenvolvimento da área. A maior parte dos trabalhos e modelos propostos surgem no contexto da visão computacional, uma vez que as arquiteturas baseadas em redes neurais são o estado de arte para muitos problemas desta área, como reconhecimento de imagem [20], transferência de estilo [57], super-resolução [56], tradução imagem para imagem [14], geração e reconstrução de imagem [1]. No entanto, as redes neurais, devido à sua capacidade de processar qualquer tipo de informação multi-dimensional, podem ser aplicadas em muitos outros contextos como, por exemplo, computação gráfica [33], biologia [41] e áudio [16, 52]. Este sucesso das redes neurais é potenciado por um desenvolvimento paralelo de *hardware*, como placas gráficas com elevada computabilidade, que permite a exploração de arquitecturas mais profundas, ou seja, com mais estágios e camadas. Isto é de especial importância, pois redes neurais mais profundas são capazes de lidar com informação e tarefas mais complexas.

A construção de um modelo generativo de imagens fotorealistas é um dos problemas fundamentais da visão computacional [8] e, só nos últimos 10 anos, surgiram modelos eficazes [51, 39, 15], devido aos avanços científicos na área das redes neurais. Estes modelos têm resultados de estado de arte para a geração de imagens fotorealistas dos mais variados sujeitos, como retratos humanos, de animais, edíficios, mapas, etc. [40, 1], e mostram-se igualmente eficazes para a geração de texto e áudio [10, 52]. Isto faz com que haja um número crescente de artistas a usar os códigos desenvolvidos pelas equipas científicas, para produzir objetos mais artísticos e narrativos do que aqueles produzidos pelos cientistas. No entanto, existem cientistas que produziram testes e resultados[1] com um grande valor estético [9, 38], porém ausentes de intenção artística, consistindo na observação direta do modelo a operar.

[1] Alguns desses resultados podem ser vistos no seguinte vídeo:
https://www.youtube.com/watch?v=ePUlJMtclcY&list=PL5278ezwmoxQEuFSbNzTMxM7McMIqx_HS

A exploração de redes neurais como sistema para gerar uma obra de arte, ainda é prematura devido, em parte, pelas dificuldades técnicas que a sua utilização representa para a maioria dos artistas. Mas, se por um lado, esta prematuridade provoca a existência de muitos problemas relativos à sua utilização, por outro deixa as possibilidades desta ferramenta num fundo misterioso e com enorme potencial. Algumas destas possibilidades têm vindo a ser introduzidas por artistas de uma forma quase experimental, onde "a obra" passa a ser um conjunto de ensaios, por vezes com poucas variações entre si. O processo criativo torna-se, assim, uma investigação de como os modelos generativos agem, i.e o estabelecimento de relações de causa-efeito entre os hiperparâmetros, dados de treino e os resultados finais, com o intuito de produzir algo com interesse artístico. Os trabalhos de Mario Klingemann e Robert Barrat são exemplos deste tipo de produção artística (exemplos ilustrados na fig. 3.1). Estes artistas produzem ensaios artísticos visuais e publicam-nos com grande periodicidade no *Twitter*. A cronologia destes ensaios reflete-se nos avanços dos modelos neurais na área da visão computacional, passando pelos modelos de transferência de estilo entre imagens, modelos de geração e, mais recentemente, modelos de super-resolução, previsão do próximo *frame* e tradução imagem para imagem. Ainda na geração de imagem, destaca-se Mike Tyka, pela criação das primeiras imagens de alta resolução com o *DeepDream*[2] (fig. 3.1 canto inferior direito), e pelo desenvolvimento de um modelo, baseado em redes neurais, para a geração de uma série de retratos de alta resolução, intitulada *Neural Portraits* (um desses retratos está ilustrado na fig. 3.1 canto inferior esquerdo). Como já foi referido, as redes neurais generativas não estão confinadas à geração de imagem; muito recentemente o artista computacional Memo Atken, publicou alguns ensaios artísticos de espetogramas[3] criados com redes neurais, que resultam em faixas sonoras coerentes. Na vertente da escrita, Oscar Sharp realizou um filme nomeado *Sunspring*[4], cujo guião foi totalmente gerado por um rede neural desenvolvida pelo investigador Ross Goodwin. Todos estes exemplos mostram-nos a versatilidade e diversidade de resultados das redes neurais no contexto da arte generativa.

As redes neurais, usadas pelos artistas, são treinadas com algoritmos de *reinforcement learning* e são classificadas como um tipo de agentes inteligentes (AI), pois percecionam a informação do ambiente e tomam ações que maximizam a chance de terem sucesso na tarefa para o qual foram desenhadas. Note-se que este agente inteligente não se trata de uma entidade que duplica artificialmente a inteligência humana, mas apenas que tenta simular partes cognitivas muito específicas do humano como o processo de aprendizagem e de resolução de problemas. John Searle na década de oitenta, de modo a refutar a visão computacionalista de mente, defendeu que uma entidade que faça apenas manipulação sintática não é capaz de produzir conteúdo semântico[43]. É importante realçar neste ponto, que a crítica de Searle aplica-se diretamente às redes neurais, uma vez o processo de aprendizagem de um rede é puramente uma manipulação sintática. Esta manipulação passa apenas por estabelecer um *feedback* tratável entre a informação do meio e as

[2]Programa desenvolvido por Alexander Mordvintsev para a *Google*, que faz transformação de uma imagem a partir de características extraídas com um rede neural. Na sec. 1.3 o algoritmo é apresentado em detalhe.

[3]Esses espetogramas foram publicados na página do *Twitter* do autor nos links: https://twitter.com/memotv/status/998925199012782080 e https://twitter.com/memotv/status/998758968242704385

[4]O filme está disponível no *youtube* no link: https://www.youtube.com/watch?v=LY7x2Ihqjmc

Figura 1.1: Canto superior esquerdo: Obra de Mario Klingemann gerada com um modelo neural de tradução imagem para imagem. Sem título, 2017. Canto superior direito: Obra de Robbie Barrat produzida com um modelo neural generativo. Sem título, 2017. Canto inferior esquerdo: Obra de Mike Tyka produzida com um modelo neural generativo. *Digital, Neural Portraits, 2017*. Canto inferior direito: Obra de Mike Tyka, produzida com o algoritmo *DeepDreams*. *Here Was The Final Blind Hour*, 2016.

várias camadas da rede, de modo a atualizar serialmente os seus parâmetros em acordância com a tarefa que tem que executar[35]. Neste sentido, as redes neurais apenas simulam o processo de aprendizagem. No entanto, não podemos descartar a ideia que esta capacidade de aprendizagem das redes neurais, apesar das suas lacunas, faz com que o artista computacional tenha um sistema capaz de aprender a fazer julgamentos (através de classificações) e fazer criações a partir destes. Hoje entendemos a arte generativa como qualquer prática artística onde o artista usa um sistema que é posto a funcionar, com algum grau de autonomia, contribuindo para ou gerando totalmente uma obra artística [12, 2]. Admitindo essa definição de arte generativa e se considerarmos que o sistema em causa é uma rede neural então imediatamente surgem questões como:

1. Qual é o tipo de autonomia de um sistema baseado em redes neurais?

2. Um sistema baseado em redes neurais é mais autónomo do que os restantes sistemas usados na arte generativa?

3. Qual é o tipo de resultados que se espera obter de um modelo neural?

Para responder a estas questões, torna-se necessário construir um contexto destes sistemas dentro do escopo da arte generativa computacional, de modo a perceber as semelhanças e diferenças que têm em relação a outros sistemas, como por exemplo, L-sistemas[5], autómatos celulares[6], sistemas de reação-difusão[7], etc.

1.2 Construção de um contexto das redes neurais na arte generativa

O elemento central na arte generativa é o sistema ao qual o artista cede um controlo parcial ou total sobre a obra [12], uma vez que este sistema tem implicações profundas sobre a estética geral desta, i.e. o aspeto e a relação entre os elementos que a compõe. Por exemplo, um artista generativo mais experiente identifica facilmente as obras geradas com L-sistemas, uma vez que estes sistemas conduzem a resultados com uma relação entre os elementos muito idêntica entre si. O mesmo acontece com as obras geradas através de um sistema fratal. A diversidade de resultados de cada sistema pode ser imensa, mas as regras intrinsecas às quais um dado sistema está sujeito, fazem com que os resultados compartilhem caraterísticas fundamentais.

[5] O L-sistema ou sistema de Lindenmayer é um sistema de reescrita paralela e um tipo de gramática formal. Um L-sistema consiste num alfabeto de símbolos que podem ser usados para gerar um cadeia de carateres, numa grupo de regras de produção que expandem cada símbolo numa maior cadeia de símbolos, numa cadeia de carateres que constitui o axioma inicial e, por fim, um mecanismo de tradução dos símbolos em figuras geométricas.

[6] Um autómato celular consiste numa grelha de células, cada uma com um número finito de estados. Para cada célula, são definidas um grupo de células vizinhas. Um estado inicial é definido para tempo $t = 0$. Para cada iteração no tempo t o sistema evoluiu a partir do estado inicial através de um regra fixa (geralmente, uma função matemática), que determina o novo estado de cada célula e das células vizinhas relativas.

[7] Sistemas de reação-difusão são modelos matemáticos que descrevem vários fenómenos físicos e químicos, tais como a mudança no tempo e no espaço da concentração de uma ou mais substâncias químicas, reações químicas locais nas quais as substâncias são transformadas noutras e difusão de substâncias, causando a sua dispersão numa dada superfície ou volume. Estes modelos são aplicados na arte generativa para dar comportamentos às partículas ou elementos que constituem o sistema generativo.

Margaret Boden [2] identifica dois tipos de abordagens na construção dos sistemas usados na arte generativa computacional: sistemas algorítmicos (também designados por sistemas passo-por-passo) e sistemas de regras. Os sistemas algorítmicos são sistemas cuja evolução é explicitamente descrita pelo programador, numa sequência de passos. Os sistemas de regras são sistemas cuja evolução é condicionada por um conjunto de limitações impostas pelo programador, como condições fronteira, ligações ou constrangimentos. O programador geralmente não consegue prever as consequências das regras que impõe à evolução do sistema, que, muitas vezes, entra em regimes não-lineares e caóticos. Por este motivo, parece que este tipo de sistemas têm maior autonomia relativamente às decisões conscientes do programador [2]. No entanto esta autonomia é apenas perceptual e deve-se ao facto do programador não conseguir prever as implicações de todas as regras que impõe. De facto a evolução do sistema, para um dado conjunto de regras e condições iniciais, é determinística, no entanto é, em geral, imprevisível para o programador, uma vez que este não consegue fazer o elevado número de cálculos necessários para prever todos os estados futuros do sistema. Deste modo a autonomia de um sistema entende-se a partir da previsibilidade do sistema para o programador. Esta previsibilidade dos sistemas está diretamente relacionada com a natureza das regras ou passos que o programador impõe ao sistema. Tudo isto nos indica que, para percebermos o grau de autonomia dos sistemas, neste caso baseados em redes neurais, assim como o tipo de resultados que estamos à espera de obter destes, temos que identificar primeiramente o tipo de regras a que estão sujeitos. Para isso, é necessário introduzir brevemente o que é uma rede neural e o algoritmo de retro-propagação [42] (uma apresentação mais completa das redes neurais será feita na próxima secção, com introdução de arquiteturas recentes).

Uma rede neural artificial é um tipo de rede constituída por nodos designados por neurónios artificiais. Estes neurónios artificiais são um modelo computacional inspirado nos neurónios naturais. Num neurónio natural, os sinais são recebidos atráves das sinapses que se situam na extremidade das dendrites. Se esses sinais forem suficientemente fortes, ou seja, acima de um dado *threshold*, o neurónio é ativado e emite um sinal pelo axónio. O sinal é então recebido por outras sinapses de neurónios vizinhos, podendo ser ativados ou não. O modelo computacional dos neurónios é uma representação artificial e simplicada deste processo. O modelo consiste em valores de entrada, que são multiplicados pelos respectivos pesos, que representam o quão forte é cada valor de entrada. Posteriormente, os diferentes sinais são somados (função de transferência) e encaminhados para uma função de ativação que os processa para gerar um valor de saída que, em geral, depende de um *threshold*. Uma rede neural resulta da combinação destas unidades para processar informação e simula, de um modo simplificado, a ativação de um neurónio natural. A fig. 1.2 esquematiza o processo de passagem da informação através de um neurónio artificial com uma função de ativação de *Heaviside*. Esta função de ativação representa o caso mais simples, devolvendo 0 para o caso do valor de entrada ser menor do que zero e devolvendo 1 para o caso de ser maior ou igual a 0. O valor de entrada para a função de ativação é simplesmente a soma dos valores de entrada multiplicados pelos respetivos pesos.

Vamos, então, supor que o grupo de valores de entrada ($x_1, x_2, ..., x_n$) representa um número de telefone e que queremos que o neurónio artificial devolva 1 caso esse número seja da operadora X e

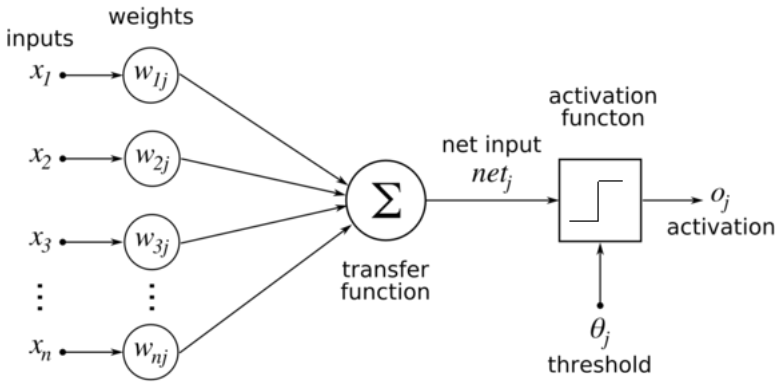


Figura 1.2: Representação de um neurónio artificial.

0 caso não seja. A única forma de fazer isso é ajustar os pesos, com valores positivos ou negativos de forma a que a soma $\sum x_i w_i$ seja maior ou igual do que zero caso seja da operadora X e menor do que zero caso não seja. Apenas nesse caso a função de ativação retornará os valores desejados. Se a quantidade de números de telefone for grande, torna-se impossível realizar esta tarefa apenas com um neurónio artificial, uma vez que os pesos produzem os resultados desejados apenas para um subconjunto de números de telefone, pois não se conseguem adaptar a todos os dados. De modo a processar mais informação, torna-se necessário interligar neurónios artificiais, numa rede designada rede neural onde os valores de saída de uma camada são os valores de entrada de outra, como esquematizado na fig. 1.3. As redes neurais com mais do que uma camada de neurónios designam-se por redes neurais profundas[8].

Para estas redes, com dezenas ou centenas de neurónios, é impossível ajustar os pesos manualmente, de modo a produzirmos os resultados desejados para toda a informação que queremos processar. Consequetemente, é necessário construir algoritmos que ajustem os pesos autonomamente; este processo designa-se por aprendizagem ou treino. O algoritmo de aprendizagem é a

[8] A profundidade de uma rede neural é determinada pelo número de camadas de neurónios intermédios, também conhecidos por *hidden layers*.

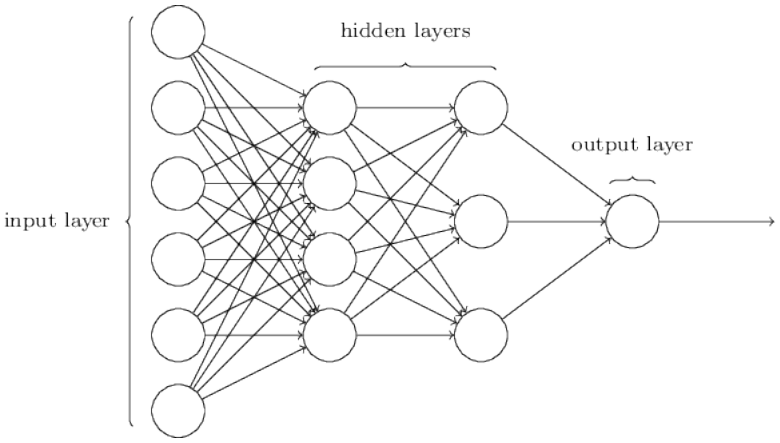


Figura 1.3: Representação de uma rede neural profunda com profundidade 2.

força motriz da rede neural, pois é o responsável por calcular as correções necessárias a cada parâmetro da rede, para que esta produza a solução desejada para os vários valores de entrada. O algoritmo de aprendizagem mais usado designa-se por algoritmo de retro-propagação. Este algoritmo foi desenhado para operar sobre redes neurais *feed-foward*, i.e, redes onde a informação flui apenas no sentido dos valores de entrada para os valores de saída, nunca passando pela mesma camada mais do que uma vez. Para estas condições o algoritmo faz passar no sentido contrário (i.e. na direção dos valores de saída para *os valores de entrada*) as correções a serem aplicadas aos parâmetros de cada camada de neurónios. Este processo é feito através de apredizagem supervisionada, que consiste em fornecer pares de valores de entrada e saída, que consistem nos valores que a rede deve receber e retornar respetivamente. Através desses pares é possível calcular um erro, através da função de perda, que consiste, por exemplo, na diferença absoluta entre o resultado atual e o expectável. Em geral, os pesos da rede são inicializados com valores aleatórios que são ajustados ao longo do tempo, através dos erros calculados, até que a diferença entre os valores de saída da rede e os valores de saída expectáveis seja mínima. Quando tal acontece, diz-se que a rede está treinada e é posteriormente testada com novos pares, para os quais não foi treinada, de modo a determinar a qualidade dos seus resultados. De uma forma sintética, a rede neural pode ser representada matematicamente como uma função P_{model} que depende dos pesos W, e que mapeia um vector com os valores de entrada $(x_1, x_2, ..., x_n)$ para um vetor de saída y (p.ex. uma imagem). Ou seja,

$$P_{model}(x_i; W) = y \tag{1.1}$$

Vamos supor que para os valores de entrada $(x_1, x_2, ..., x_n)$ queremos obter a imagem $\hat{y}$; No caso mais simples, a função de perda é apenas a diferença absoluta entre o y (imagem produzida) e $\hat{y}$ (imagem expectável), i.e.

$$L(y, \hat{y}) \propto |y - \hat{y}| \tag{1.2}$$

Seguidamente, este erro é retro-propagado pela rede, de modo a determinar os gradientes (quantidade a ser somada a cada parâmetro da rede), necessários para a minimização do erro, i.e. da função de perda L. A minimização da função L é o equivalente a aproximar a imagem produzida pela rede à imagem expectável.

Como referido anteriormente, o grau e o tipo de autonomia de qualquer sistema generativo dependem da natureza das regras ou contragimentos aos quais está sujeito. No caso de um sistema artificial neural, a minimização da função de perda L e a morfologia da rede são os constragimentos ou regras às quais o sistema tem que obedecer. A minimização da função L é a regra fundamental do sistema pois, através do algoritmo de retro-propagação, tem efeito sobre toda a rede, ditando o tipo de valores de saída que a rede deve retornar. Sem a função L, a rede produziria valores de saída com uma distribuição aleatória, que não têm grande interesse estético. Galanter, ao contextualizar a arte generativa através da teoria da complexidade [12], refere que os sistemas produzem resultados esteticamente interessantes, quando estão num estado de mistura entre ordem

e desordem, ou seja, quando apresentam complexidade efetiva maior do que zero. No processo de aprendizagem de uma rede neural, há um deslocamento do sistema da região de desordem total para uma região de mistura entre ordem e desordem. O ponto de paragem é ditado pela minimização da função L, podendo ser um ponto de maior ou menor complexidade efetiva. Isto mostra que as redes neurais podem gerar distribuições de complexidades efetivas muito diferentes (determinadas pela função L e dados de treino), e consequentemente, gerar obras (visuais, sonoras, etc.) de naturezas extremamente variadas.

Note-se que a função L não contém, explicitamente, nenhuma informação de como o sistema se deve organizar internamente, mas apenas do que deve produzir. Dito por outras palavras, tanto a função L como a morfologia da rede não impõe valores específicos ou comportamentos às unidades do sistema (i.e. aos neurónios). O sistema tem a autonomia de se organizar internamente com a obrigação de chegar a um estado final onde minimiza a função L, ou seja, onde produz resultados com distribuições próximas das distribuições reais dos dados usados para o treino da rede. Este tipo de autonomia faz das redes neurais um sistema consideralvelmente diferente dos restantes sistemas usados na arte generativa computacional. Os sistemas de partículas e genéticos, por exemplo, que são sistemas que também têm uma complexidade efetiva elevada, são governados por regras que impõe comportamentos às unidades do sistema (como comportamentos cinéticos, de destruição, de multiplicação, etc), mas que deixam livre a forma como sistema evoluiu e produz resultados, ao contrário do que acontece nos sistemas neurais. A autonomia destes sistemas está na sua evolução, por vezes caótica, chegando a um ou mais estados de equilíbrio finais, muitas vezes imprevisíveis para o programador. Todavia, um modelo neural, quando treinado para gerar retratos de pessoas, nunca produzirá uma paisagem montanhosa, pois durante o processo de treino evoluiu restritamente para um conjunto de estados finais próximos (i.e. com pequenas variações) dos retratos reais de pessoas. Nesse sentido, os resultados de um modelo neural são mais previsíveis, pelo menos teoricamente. A imprevisibilidade está na forma como a rede se organiza internamente e aprende a gerar distribuições tão complexas, como, por exemplo, de uma paisagem montanhosa. Numa perspetiva artística, esta previsibilidade dos modelos permite ao artista ensinar ao computador a gerar resultados com as composições e elementos específicos que busca para a sua obra. Todavia, esta previsibilidade pode ser limitadora se o artista quiser dar ao modelo neural uma autonomia mais radical e criativa.

A comunidade científica procura modelos que gerem resultados realistas, no entanto, um artista é livre de explorar os modelos de uma forma mais experimental, i.e. treinar as redes com dados não realistas ou consistentes entre si, usar funções de perda diferentes ou treinar uma rede para um propósito e usá-la para outro, gerando artefactos imprevisíveis. O artista consegue assim ensinar à rede neural a gerar resultados próximos daqueles que deseja obter e, simultaneamente, ceder espaço para variações imprevistas. Para finalizar esta contextualização, importa referir que uma rede neural treinada é equivalente a uma ferramenta generativa para o artista, e não apenas um resultado. A função $P_{model}(x_i;W) = y$ é contínua e contém uma infinidade de soluções. A obra final do artista resulta da exploração desta função, que pode usada para fins muito diferentes para o qual foi treinada; isto faz das redes neurais um sistema extraordinário para a arte generativa

computacional.

Este grupo de considerações responde diretamente às questões 1,2,3 da secção 1.1 e cria uma base ou contexto teórico das redes neurais como sistema de arte generativa computacional.

Na próxima secção, vão ser apresentadas as arquiteturas neurais revelantes para a arte generativa. Esta apresentação será crucial para uma análise detalhada dos tipos de uso que os artistas fazem dos modelos neurais, identificando também os problemas da sua aplicação.

1.3 Arquiteturas neurais usadas na arte generativa computacional

Antes de apresentarmos as arquiteturas propriamente ditas, devemos discutir os conceitos que motivam a sua construção. O objetivo destas arquiteturas, no contexto da arte generativa, é simular o processo artístico, ou seja, ser um artista artificial capaz de produzir obras de arte que consigam ser tão válidas quanto as reais. Deste modo, as tarefas que as redes neurais aprendem a realizar baseiam-se na ideia do processo artístico humano, i.e. nas capacidades cognitivas e físicas humanas necessárias para fazer uma obra de arte. Durante a década de 90, surgiu o conceito de artista artificial[9] [46] no ramo da arte evolucionária. Spector definiu-o como um sistema capaz de criar obras de arte esteticamente válidas, com intervenção humana mínima. Vários artistas artificiais foram desenvolvidos, como o *Aaron* do artista Harold Cohen, o algoritmo genético *Mutator* [27] de William Latham e Stephen Todd, que fazia evolução seletiva de designs de casas e frascos de perfumes e o algoritmo de Karl Sims [45] que usa funções matemáticas, codificadas na forma de *Lisp S-expressions*, gerando imagens a partir destas. Mais tarde, Penousal Machado vem identificar as fraquezas desses modelos e expandir a definição de artista artificial [34]. Para ele um artista artificial deve apresentar capacidades representacionais gerais, isto é, ser capaz de gerar qualquer imagem; por outro lado, deve conseguir aprender ao longo do tempo, como qualquer artista humano; deve de integrar conhecimento inicial, como um conjunto de obras de arte, uma vez que nenhum artista começa do zero; e por fim, deve ter a capacidade de fazer julgamentos estéticos. De modo a satisfazer estes requisitos, Penousal Machado constrói um sistema com duas componentes: a componente de geração e a componente de avaliação estética (que integra o conhecimento prévio de obras). Para a parte da geração de imagem, ele usa um algoritmo genético que vai evoluindo através de mutação e recombinação das imagens anteriores que apresentam maior valor estético. Para a componente da avaliação, usa pioneiramente, nesta área, redes neurais. A arquitetura que usou tinha apenas duas camadas de neurónios artificiais, devido às limitações do *hardware* da época. Ele relaciona cada camada com as etapas do julgamento estético, i.e. a primeira camada realiza uma leitura da unidade, predominância, variedade, balanço, continuidade, simetria, proporção e ritmo da obra e a segunda camada realiza a avaliação estética final. Hoje sabemos que duas camadas de neurónios não são capazes de fazer um julgamento eficaz para tantas caraterísticas, no entanto as arquiteturas neurais recentes para geração de imagens baseiam-se

[9] Em inglês, *constructed artist*. A tradução proposta imprega o termo artificial, uma vez que "artificial" significa algo que é produzido pelo homem; que se faz por arte ou indústria, segundo o Dicionário Infopédia da Língua Portuguesa. Na minha perspetiva, este termo captura melhor o significado da palavra inglesa "*constructed*" do que o termo, mais direto, "construído".

na abordagem de Penousal Machado[15, 36, 9, 40, 8, 21], i.e. com uma componente de classificação/avaliação (que integra o conhecimento externo) e uma componente de geração com alta capacidade representacional.

A partir deste ponto, vamos apresentar arquiteturas neurais usadas na arte generativa visual, em particular, uma vez que o presente trabalho se foca na geração de imagens. No entanto, estas arquiteturas foram usadas noutros contextos, como na geração e classificação de texto e som.

1.3.1 Classificação de imagens com redes neurais convolutivas

As arquiteturas neurais para classificação/avaliação de imagens são a base das arquitecturas neurais generativas. Esta relação entre ambas as arquiteturas é direta, uma vez que a criação de uma arquitetura capaz de reconhecer as caraterísticas das imagens, implica que seja capaz de construir uma representação numérica de cada uma das caraterísticas. Se essa extração é possível, então podemos supor que o processo inverso também é válido, i.e. a geração de imagens a partir da representação das caraterísticas que devem ter - essa é a hipótese que dá origem aos sistemas neurais generativos. Os artistas computacionais não usam frequentemente classificadores de um modo isolado, no entanto utilizam-nos indiretamente através dos modelos neurais generativos e, deste modo, estas arquiteturas fazem parte do grupo de arquiteturas neurais mais usadas pelos artistas computacionais.

Nos últimos anos, as redes neurais têm permitido uma evolução notável da visão computacional. Grande parte desse avanço deve-se, particularmente, ao sucesso das arquiteturas neurais convolutivas usadas para as tarefas discriminativas. As arquiteturas neurais baseadas em redes neurais convolutivas são o estado de arte para as tarefas de classificação de imagem [20], segmentação semântica [55], reconhecimento de ações [53], entre outros. As redes neurais convolutivas (CNN ou ConvNet) são um tipo de redes neurais artificiais *feed-forward*, compostas por uma camada de entrada, um camada de saída e camadas intermédias (geralmente >3), organizadas numa estrutura inspirada no cortéx visual. O cortéx visual contém uma estrutura complexa de células, que são sensíveis a pequenas sub-regiões do campo visual, designadas por campos recetivos. Os campos receptivos de células adjancentes são sobrepostos parcialmente de modo a cobrir eficazmente todo o campo visual. Essas células actuam como filtros na imagem que chega à retina e que vão sendo aplicados sucessivamente de modo a detectar as diferentes caraterísticas da imagem, como a cor e a correlação espacial local da imagem. No trabalho de Hubel e Wiesel's [22], são identificados dois tipos de células no cortéx visual: células mais simples que identificam padrões de arestas no seu campo recetivo de dimensão reduzida e células mais complexas, com campos recetivos maiores e que são localmente invariantes para a posição exata do padrão.

As CNNs são inspiradas nesta ideia de que neurónios com diferentes campos recetivos capturam diferentes caraterísticas. De modo a que diferentes camadas de uma CNN tenham neurónios artificiais com campos recetivos diferentes, é usado um padrão de conectividade (fig. 1.4), que consiste na conexão de subconjuntos de neurónios de uma camada, a cada neurónio da camada seguinte. Este processo faz com que o campo recetivo dos neurónios da camada seguinte sejam cada vez maiores.

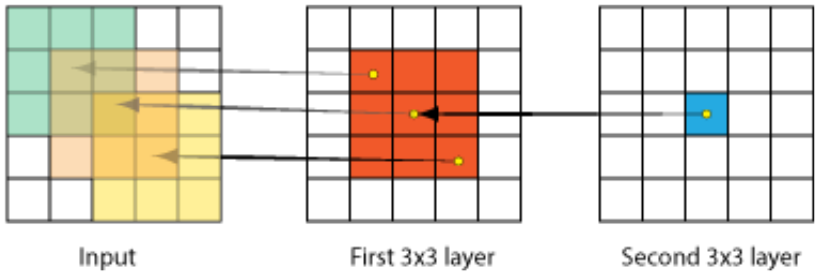


Figura 1.4: Padrão de conectividade usado entre as camadas de uma CNN.

De modo a explicitar o padrão de conectividade usado numa CNN, vamos supor que a camada de entrada é uma imagem *bitmap*, i.e, uma grelha 2D de píxeis. Seguidamente, ligamos a cada neurónio da segunda camada da rede, um quadrado de 9 píxeis dessa imagem de entrada. Cada neurónio fica assim com um campo recetivo de 3×3 píxeis. Seguidamente, são ligados a cada neurónio da terceira camada, um conjunto de 3×3 neurónios da segunda camada. Isto faz com que o campo recetivo de cada um dos neurónios da terceira camada tenha dimensão de 5×5 píxeis relativamente à imagem de entrada. Com este padrão de conectividade, a CNN tem camadas com neurónios com campos recetivos pequenos, que capturam as caraterísticas mais locais das imagens (p.ex. padrões de arestas e cor), e camadas com neurónios com campos recetivos maiores, mais sensíveis às caraterísticas globais (p.ex. conteúdo da imagem), tal como acontece nas células simples e complexas do córtex visual.

Outra das caraterísticas principais das CNN é o uso de camadas com pesos partilhados(organizados em filtros), que são replicados para o campo recetivo de todos os neurónios destas(fig. 1.5). Desta forma, a dimensão dos filtros têm assim a dimensão do campo recetivo dos neurónios da camada.

Cada filtro representa uma caraterística específica. Os neurónios, que avaliam esse filtro, são ativados se essa caraterística estiver presente nos seus campos recetivos. A avaliação de determinado filtro nos campos recetivos de todos os neurónios da camada, dá origem a um mapa geral de ativações, designado por *feature map*. Como à medida que a camada posterior tem neurónios com um campo recetivo maior, então os *features maps* gerados por estas capturam caraterísticas mais globais, enquanto que as camadas iniciais, com neurónios com campos receptivos mais reduzidos, capturam carateristicas mais locais. Isto é demonstrado no trabalho de Matthew D. Zeiler and Rob Fergus [54] onde está publicada a figura 1.6. Nessa figura vemos que as camadas 1, 2 e 3 têm *feature maps* que capturam caraterísticas mais locais da imagem, como padrões de arestas e cor. As camadas 4 e 5, têm *features maps* que capturam caraterísticas mais gerais como a composição e o contéudo da imagem.

As camadas que possuem estas duas caraterísticas, i.e. pesos partilhados e o padrão de conectividade descrito anteriormente, são designadas por *convolutional layers* e são o elemento principal das arquiteturas CNN. As camadas como *fully-connected layer*, *pooling layer* e *rectification layer*

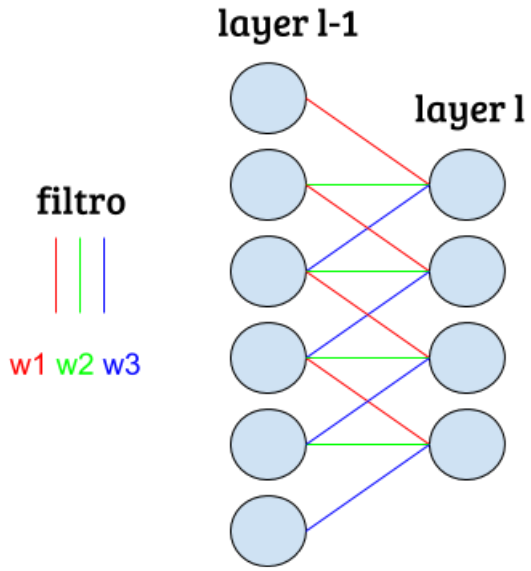


Figura 1.5: Representação do uso de pesos partilhados ($w1$, $w2$, $w3$) para todos os neurónios de dada camada l.

são, juntamente com as *convolutional layers*, as camadas mais comuns utilizadas nas arquitecturas CNN. Segue-se uma breve descrição de cada uma delas. Para os leitores que não estão familiarizados com estas camadas, encontra-se um descrição mais detalhada destas no Anexo A.

Convolutional Layers (CONV-layer): Estas camadas são compostas por um conjunto de filtros treináveis, representados por matrizes de valores reais e que são interpretados como pesos que são partilhados por todos os neurónios da camada. Estes filtros são convoluídos com os dados de entrada presentes no campo receptivo de cada neurónio, determinando a sua ativação. Este processo de convolução está descrito na figura 1.7 e consiste no produto escalar dos valores da matriz do filtro com os valores do campo receptivo do neurónio em causa. O conjunto dos valores de todas as activações, para um dado filtro, designa-se por *feature map*. O *feature map* indica as regiões nas quais a característica específica relativa ao filtro é encontrada. Para estas camadas, o algoritmo de retro-propagação ajusta os parâmetros de cada filtro, de modo a serem adequados para avaliação das características relevantes para a classificação das imagens. O uso de pesos partilhados (filtros) aumenta a eficiência do treino destas redes, uma vez que reduz o número de parâmetros livres (tipicamente diferentes para cada neurónio).

Fully-Connect Layers (FC-layer): Nestas camadas os neurónios estão conectados a todo o volume de entrada, ao contrário do que acontece nos *convolutional layers*. Estas camadas são usadas tipicamente no fim das arquitecturas CNN, para proceder à classificação final, uma vez que cada neurónio tem acesso a todas as ativações presentes na camada anterior.

Figura 1.6: Visualização dos *feature maps* das diferentes camadas de uma CNN. Imagem retirada do artigo [54].

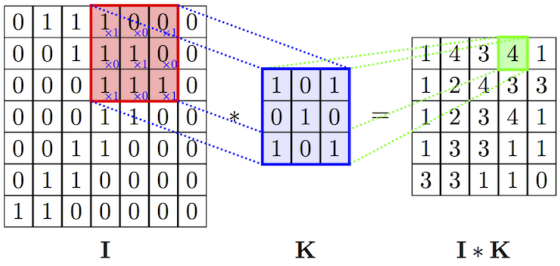


Figura 1.7: Convolução de um filtro K com uma imagem I. Esse processo dá origem a um *feature map* $I * K$, que representa o conjunto das ativações dos neurónios da camada convolutiva.

Pooling Layers (POOL-layer): Estas camadas são utilizadas para reduzir a dimensão espacial do volume de saída produzido pelos *convolutional layers*. De modo a fazer a redução, estas camadas reduzem subregiões do volume de entrada a um valor apenas. Por exemplo, o *maxpooling layer* reduz cada uma das subregiões ao valor máximo contido nesta, como representado na figura 1.8.

Rectification Layers (Re-layer): Esta camada aplica uma função de ativação ao volume de entrada. O exemplo mais conhecido é o ReLU-layer, que aplica a função de rampa $f(x) = \max(0, x)$ ao volume de entrada, retendo apenas com os valores positivos das ativações da camada anterior. Estas camadas são úteis para quebrar a linearidade do sistema, reduzindo a probabilidade de sobreajuste da rede aos dados[10].

1.3.2 Arquiteturas CNN mais usadas para a classificação de imagens e extração de *feature maps*

O avanço do *hardware* ao longo dos anos, permitiu treinar arquitecturas CNN cada vez mais profundas, i.e. com um elevado número de camadas intermédias. A original *LeNet5* [29], tem 5

[10]Em inglês, este fenómeno é conhecido por *overfitting* e acontece quando a rede se ajusta demasiadamente aos dados de treino, não conseguindo generalizar os seus parâmetros, de modo a processar com sucesso outros dados.

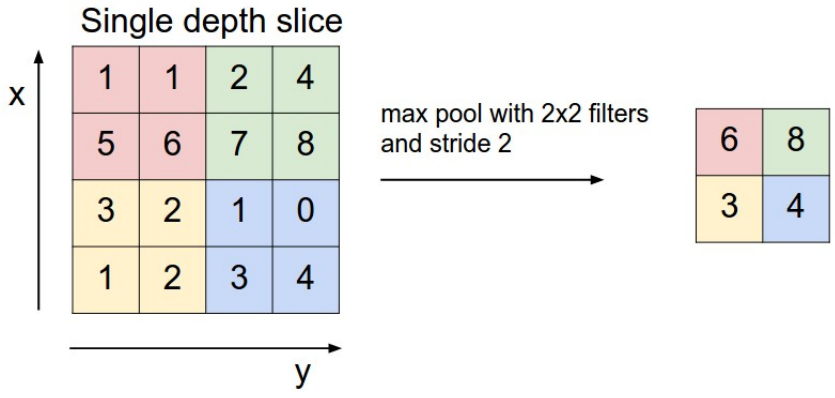


Figura 1.8: Operação da camada de *maxpooling* sobre o volume de entrada.

camadas intermédias, conseguindo lidar com a classificação de imagens de bibliotecas simples, como a MNIST [30], com 60.000 imagens de resolução baixa (32×32 px). No entanto, a introdução das *VGG-nets* [44] em 2014, com 19 camadas, permitiram pela primeira vez lidar com bibliotecas como a *ImageNet* [6] com 15 milhões de imagens de resolução alta classificadas em 22 milhares de categorias. Seguidamente, as *Inception-v3 nets* [49], *highway nets* [47] e as *residual nets* [18] ultrapassaram a barreira das 50, 100 e 1000 camadas, respetivamente. As arquiteturas CNN mais profundas, permitem lidar com bibliotecas mais complexas, no entanto são mais difíceis de treinar, uma vez que os gradientes, quando retro-propagados por tantas camadas, tendem a desaparecer. Este problema é combatido com o desenvolvimento de micro-arquiteturas (discutidas no Anexo B) que são anexadas à convencional arquitectura *end-to-end*, de modo a que os gradientes se preservem até ao ínicio da rede. Todavia, convém salientar que as arquiteturas mais profundas, não são necessariamente mais precisas do que as menos profundas. Tal depende da biblioteca com a qual a rede é treinada e das arquiteturas e micro-arquiteturas que são usadas [20].

Como foi referido anteriormente, os artistas computacionais não usam frequentemente arquiteturas de classificação de imagem de um modo isolado, mas sim de um modo indireto, através dos modelos generativos, como por exemplo, modelos para transferência de estilo entre imagens ou geração. Estes modelos generativos usam arquiteturas CNN para fazer a extração dos *features maps* das imagens de treino, de modo a gerar imagens com as mesmas caraterísticas destas. Nestes modelos, a extração dos *feature maps* é feita, geralmente, com *VGG-nets*, *Standard Res-net* e *InceptionV3-net* (apresentadas no Anexo B) por serem mais simples e com uma eficácia comparável às das arquiteturas mais complexas. Irei agora apresentar os modelos neurais que os artistas computacionais utilizam de um modo direto.

1.3.3 DeepDreams original e o algoritmo de transferência de estilo

As arquiteturas apresentadas na secção anterior, por si só, não são capazes de gerar imagens, uma vez que são desenhadas para classificação. No entanto, essas arquiteturas são capazes de extrair as caraterísticas das imagens através da geração de *features maps*; este processo é a base dos modelos neurais generativos.

O aparecimento consistente de modelos baseados em CNNs para a geração de arte visual computacional deu-se em 2014, quando Alex Mordvintsev cria o *DeepDreams* e Leon Gatys publica o algoritmo de transferência de estilo [13]. Ambos os modelos fazem uso da potencialidade dos *features maps* extraídos de imagens, através de CNNs (p.ex. a *Inception net* e *VGG-19*), para gerar novas imagens baseadas nestes. Deste modo, as imagens geradas pelos modelos generativos, contém as caraterísticas que são identificadas nos *feature maps* extraídos das imagens de treino.

DeepDreams: a visão interna de uma CNN

A ideia base do algoritmo do *DeepDreams* é a visualização das caraterísticas que uma CNN identifica numa imagem através da amplicação destas. A imagem em causa é transformada de modo a amplificar as características que uma ou mais camadas da CNN identifica nela e que estão

Figura 1.9: Obras resultantes da exploração do DeepDreams. À esquerda a obra "Jurogumo" de Mike Tyka. À direita um obra sem título de Gene Kogan..

representadas nos respectivos *feature maps*. Consideremos, por exemplo, que a imagem a ser transformada representa uma paisagem e que a CNN usada está treinada para reconhecer imagens de animais. Neste caso, a CNN irá identificar na imagem da paisagem, padrões semelhantes aos padrões que usa para classificar as imagens de animais. Caso algum padrão seja encontrado, o programa amplifica-o sucessivamente até que fique reconhecível. Por este motivo os resultados finais tendem a parecer psicadélicos. Os detalhes do algoritmo encontram-se no anexo C.

O *DeepDreams* foi um dos primeiros passos na aplicação das redes neurais para gerar obras de arte. Alguns artistas, como Mike Tyka e Gene Kogan, exploraram este programa logo após o seu lançamento para a geração de imagens como as representadas na figura 1.9. No entanto, a pouca diversidade de resultados e a massificação da ferramenta nas comunidades *online*, depressa fez com que esta técnica caísse em desuso na comunidade artística.

Após poucos meses, Gatys publica um algoritmo de transferência de estilo entre imagens, que faz um uso mais abrangente dos *features maps* gerados pelas CNNs, dando-lhes uma interpretação.

Transferência de estilo com CNNs

O algoritmo de transferência de estilo proposto de Gatys [13], vem resolver o problema complexo da renderização do conteúdo semântico de uma imagem com o estilo de outra. Até então, os outros métodos falhavam pela falta de uma representação exclusiva do conteúdo de uma imagem e uma representação exclusiva do estilo da outra imagem, de modo a que uma recombinação fosse possível. Gatys propõe um algoritmo que usa uma rede neural convolutiva para extrair as diferentes representações da imagem, conseguindo separar a informação relativa ao conteúdo da informação relativa ao estilo. Essa informação é depois usada para gerar uma nova imagem que combina o conteúdo de uma imagem arbitrária com a aparência/estilo de outra (como p.ex o conteúdo de uma fotografia combinado com o estilo dos quadros de Van Gogh).

Numa CNN quanto mais profunda for uma camada convolucional, maior é o campo receptivo dos seus neurónios relativamente à imagem original; consequentemente, os *features maps* gerados por estas camadas são mais sensíveis ao conteúdo geral da imagem e relativamente insensíveis ao

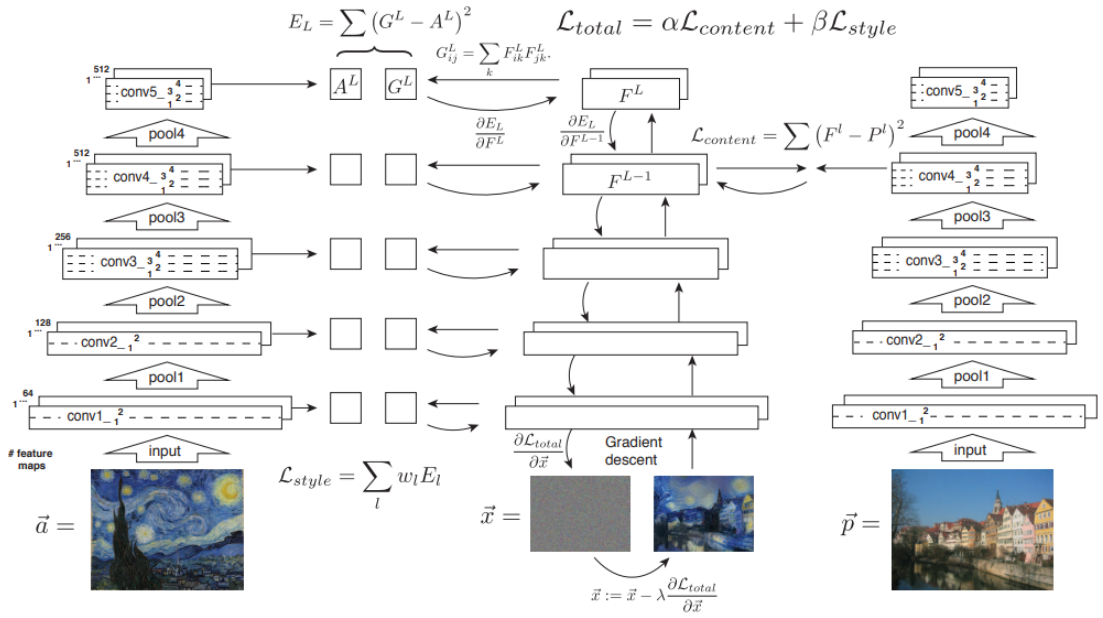


Figura 1.10: Esquema do algoritmo de transferência de estilo. Fonte da imagem: [13]

estilo desta (fig. 1.6). Já as camadas iniciais, com neurónios com campos recetivos de poucos píxeis, são mais sensíveis ao estilo e padrões locais, não tendo acesso a qualquer informação explícita do conteúdo da imagem. Por este motivo, Gatys controí a representação do conteúdo[11] de uma imagem a partir dos *feature maps* gerados pelas camadas mais profundas de uma CNN. Para obter a representação do estilo[12], Gatys usa um *feature space*, que consiste na composição linear dos *feature maps* consigo próprios, de modo a descartar qualquer informação relativa à organização espacial da imagem, uma vez que esta não interessa para a representação do estilo. O algoritmo usa a representação do conteúdo de um imagem A e a representação do estilo de uma imagem B para gerar uma imagem através de uma CNN que força a imagem gerada a ter o conteúdo da imagem A e o estilo da imagem B (como representado na fig. 1.10). Os detalhes do algoritmo são apresentados no Anexo D.

Este algoritmo resolveu o problema complexo de transferência da estilo entre imagens e, simultâneamente, deu uma interpretação aos *feature maps* produzidos pelas camadas de uma CNN. As duas representações introduzidas são de enorme relevância para a área da visão computacional e são extremamente uteís para o desenvolvimento de modelos para a arte computacional.

Em 2016, Johnson, Alahi e Fei apresentaram a versão otimizada deste algoritmo para aplicação em tempo real [25]. O artista Gene Kogan usa esta versão otimizada para fazer a instalação *Cubist Mirror* (fig. 1.11 lado direito), que renderiza em tempo real o conteúdo dos *frames* da câmera com um estilo pictórico. Pouco tempo depois, surgiram as aplicações para telemóvel que faziam essencialmente o mesmo que a instalação de Gene Kogan, massificando o algoritmo (a fig. 1.11, lado esquerdo, mostra uma dessas aplicações).

[11] Apresentado em detalhe no Anexo C.
[12] Apresentado em detalhe no Anexo C.

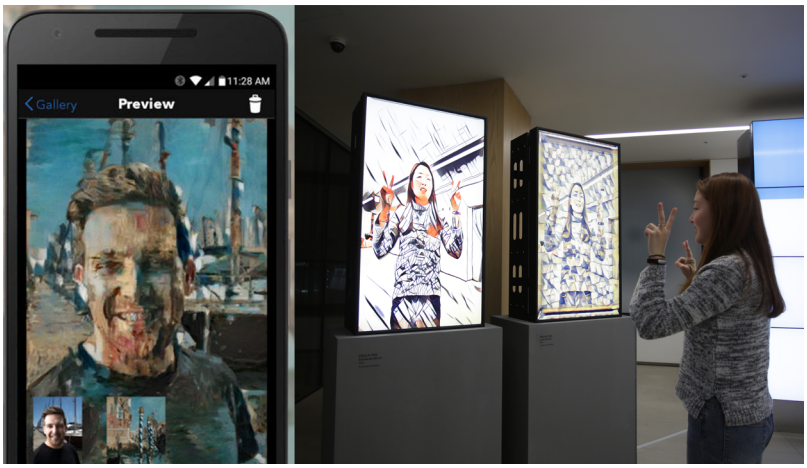


Figura 1.11: Lado esquerdo: Fotografia (*selfie*) manipulada com uma aplicação para transferência de estilo entre imagens chamada *Pikazo*. Lado direito: Fotografia da instalação *Cubist Mirror* (2016), do artista Gene Kogan.

A maioria das aplicações de transferência de estilo têm modelos estáticos, limitadas a um certo número de estilos e com a mesma interpretação das imagens iniciais. Para uma exploração profunda deste método, os artistas têm que experimentar outras funções de perda e fazer a extração dos *feature maps* das imagens iniciais com CNNs diferentes, por vezes mal treinadas, de modo a conseguir resultados inesperados. Uma exploração virtuosa deste algoritmo é feita pelo Jamie Dutcher (director de efeitos especiais), no videoclip *When You Die*[13] realizado por Mike Burakoff e Hallie Cooper-Novack em 2017.

1.3.4 Modelos Neurais Generativos

Como vimos anteriormente, os modelos discrimativos CNN são extremamente úteis para extrair representações das caraterísticas das imagens, todavia não conseguem capturar a sua distribuição. Os modelos generativos são modelos capazes de capturar a distribuição complexa de um conjunto de imagens. Uma vez obtida esta distribuição, o modelo é capaz de gerar imagens percetualmente iguais às imagens desse conjunto. As redes neurais são aproximadores universais[14] e, consequentemente, são candidatos naturais à construção de modelos generativos. Todavia, os modelos neurais generativos não são tão eficazes quanto os modelos neurais discriminativos, no que toca ao processamento de imagens de diferentes categorias. Na prática, ainda só é possível treinar modelos neurais generativos para gerar imagens da mesma categoria, como por exemplo, da categoria "casas", "carros", "retratos", etc. [1, 40].

Muitos esforços foram feitos de modo a produzir modelos neurais generativos eficientes. Os

[13] O vídeo pode ser visualizado no *link*: https://www.youtube.com/watch?v=tmozGmGoJuw
[14] As condições desta proposição são apresentadas no teorema da aproximação universal.

primeiros modelos incluíam máquinas de Boltzmann restritas[15] e *deep belief networks*[16], no entanto produziam resultados muito limitados. Apenas recentemente surgiram modelos mais eficazes tais como auto-codificadores variacionais[17] [39] e modelos neurais recurrentes[18] [51]. No entanto, esses modelos falham, respetivamente, na convergência para a distribuição das imagens de treino[19] e geração de imagens com médias e altas resoluções[20]. Em 2014, Ian Goodfellow propõe um modelo generativo, designado por redes neurais adversariais (GANs) [15], que resolve parcialmente estes problemas, sendo, desde então, o modelo neural generativo mais usado. Os modelos baseados em GANs que têm sido propostos ao longo dos últimos anos são o estado de arte para tarefas como tradução imagem para imagem [57], super-resolução [5] e geração de imagem a partir de texto [3]. Estes modelos têm sido o foco recente dos artistas computacionais. Os modelos propostos no presente trabalho também usam GANs, sendo crucial fazer uma introdução destas arquitecturas. Uma apresentação mais detalhada é feita no Anexo E.

Redes neurais adversariais

As redes neurais adversariais (GANs) são compostas por dois modelos, designados por gerador e discriminador, que são treinados alternadamente para competir um com o outro. O discriminador D é otimizado para distinguir as imagens reais das imagens produzidas pelo gerador G. Enquanto isto, o gerador G é otimizado gerar imagens que sejam difíceis para o discriminador diferenciar das imagens de treino. No geral, o processo de treino do gerador e discriminador é similar a um jogo minimax entre dois jogadores[21] com a seguinte função objectivo:

$$\min_G \max_D V(D,G) = \min_G \max_D E_{x\sim p_{data}(x)}[\log D(x)] + E_{x\sim p(z)}[\log(1-D(G(z)))] \tag{1.3}$$

onde x é a imagem real com a distribuição dos dados reais p_{data} e z é um vetor aleatório da distribuição p_z (p.ex. a distribuição de *Gauss*). Este modelo e a respetiva função objetivo são explicados com mais detalhe no Anexo E.

[15] Máquinas de Boltzmann restritas são uma rede neural generativa estocástica que pode aprender a distruibuição de probabilidade dos dados de entrada.

[16] Redes de crença profunda é um tipo de rede neural profunda, composta por multíplas camadas de variáveis latentes, com conexões entre camadas, sem estabelecer conexões entre nodos da mesma camada.

[17] Tipo de rede neural artificial usada para fazer codificações eficientes de dados com treino não-supervisionado e que faz proposições fortes relativamente à distribuição do seu espaço latente.

[18] Tipo de redes neurais artificiais onde as conexões entre neurónios formam um grafo direto ao longo de uma sequência. Ao contrário das redes neurais *feed-foward*, podem usar o seu estado interno (memória) para processar sequências de valores de entradas, tais como sequências de caracteres.

[19] Os auto-codificadores variacionais são ineficientes na aproximação da distribuição posterior, pois existe um *gap* entre a função de perda e a verdadeira função de verossimilhança; isto faz com que as imagens geradas tenham distribuições diferentes das imagens de treino.

[20] Os modelos neurais generativos recurrentes[51] usam unidades de *Long short-term memory (LSTM)* que são computacionalmente pesadas, fazendo deste método pouco escalável para imagens de alta resolução.

[21] Jogo entre dois jogadores, no qual um deles está a maximizar as suas chances de ganhar, enquanto que o outro está a minimizar as chances disso acontecer (maximizando indirectamente a chances dele próprio ganhar). O primeiro jogador deve escolher a jogada com maior pontuação dentro do conjunto das jogadas com menores pontuações que o segundo jogador o força a ter.

A rede neural adversarial condicionada (cGANs) [36] é uma extensão da GAN, onde o gerador G e o discriminador D recebem uma variável adicional, levando à transformação de $G(z)$ em $G(z,c)$ e $D(x)$ em $D(x,c)$. Esta formulação permite que G gere imagens condicionadas pelas variáveis c. A função objetivo, neste caso, resulta da substituição direta dos modelos G e D reformulados pelos modelos G e D presentes na equação 1.3, dando origem à equação

$$\min_G \max_D V(D,G) = \min_G \max_D E_{x\sim p(x)}[\log D(x|c)] + E_{x\sim p(z)}[\log(1-D(G(z|c)))] \tag{1.4}$$

Este modelo e a respetiva função objetivo são explicados com mais detalhe no Anexo E.

O gerador e o discriminador são modelos que consistem em redes neurais profundas. Em 2016, Radford e Metz propõem uma arquitetura estável para o gerador e discriminador baseada em camadas convolutivas, chamada *Deep Convolutional GAN* (DCGAN) [40]. Esta arquitetura serve, actualmente, de base para imensos trabalhos científicos que usam treino adversarial para resolver problemas da visão computacional. Os modelos baseados em DCGANs também são os modelos generativos mais usados pela comunidade artística, onde estão incluídos, por exemplo, os artistas Mario Klingemann, Robbie Barrat e Memo Atken.

Uma das propriedades mais importantes das GANs, em especial para a arte generativa, é a linearidade do espaço representacional como demonstrado por Radford[40]. Consideremos que temos uma $G(z)$ treinada para gerar retratos humanos e um grupo de vetores z_{man} que geram retratos de homens, um grupo de vetores $z_{man-glasses}$ que geram retratos de homens com óculos e um grupo de vetores z_{woman} que geram retratos de mulheres, então por linearidade do espaço representacional a combinação aritmética

$$z_{man-glasses} - z_{man} + z_{woman} = z_{woman-glasses} \tag{1.5}$$

dá origem a um grupo de vetores $z_{woman-glasses}$ que geram retratos de mulheres com óculos. A linearidade do espaço representacional, permite fazer transições suaves entre imagens geradas por $G(z)$, através de interpolação entre valores de z. Isto faz da função $G(z)$ uma ferramenta generativa poderosa para os artistas computacionais, uma vez que para um passo suficientemente pequeno dz, $G(z+dz)$ representa uma imagem com variações pequenas da imagem $G(z)$, permitindo a criação de vídeos coerentes através de "viagens" lentas no espaço latente z.

Modelos adversariais usados na comunidade dos artistas computacionais

Nos últimos dois anos, os artistas computacionais têm explorado com grande sucesso os modelos generativos adversariais disponibilizados pelos cientistas nas plataformas como o *Github*. A escolha do modelo depende das intenções do artista. Os modelos como DCGANs ou vanillaGANs (governados pela eq.1.3) são usados quando o artista pretende fazer uma geração de imagens com a mesma distribuição das imagens de treino, mas sem estar condicionada por nenhum tipo de informação adicional. Robbie Barrat[22] usou DCGANs para gerar paisagens (fig.1.12 lado direito) e,

[22]Os seus ensaios artísticos são publicados em https://twitter.com/drbeef_

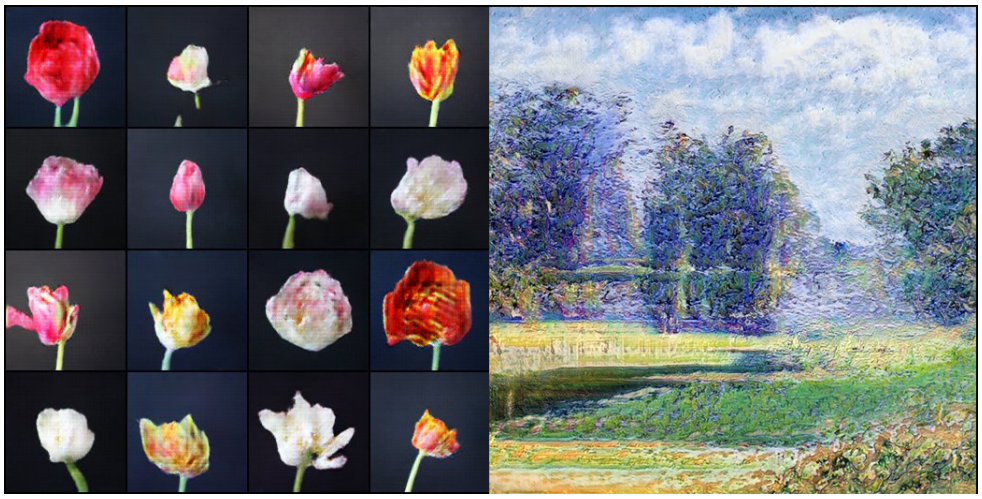


Figura 1.12: Obras geradas com DCGANs. Lado esquerdo: Uma série de tulipas artificiais da artista Anna Ridler. Sem título, 2018. Lado direito: Uma paisagem artificial do artista Robbie Barrat. Sem título, 2017.

mais recentemente, para gerar retratos de corpos nus. Os seus dados de treino são, frequentemente, compostos por quadros de artistas, dando origem a ensaios artísticos com uma estética de pintura a óleo. As composições das imagens geradas, por vezes surreais, indicam que o Robbie Barrat usa dados de treino com imagens que têm grandes variações entre si. Recentemente a capa *online* da *Bloomberg Businessweek* incorporou um vídeo feito por este artista, que representa uma "viagem" no espaço latente de uma DCGAN treinada para gerar paisagens. A artista computacional Anna Ridler[23] criou, recentemente, uma coleção de 10,000 fotografias de tulipas e usou-a para treinar uma DCGAN a gerar novas espécies (fig.1.12 lado esquerdo). Posteriormente, a artista selecionou as espécies que mais gostava, fez as impressões das imagens e catalogou-as.

Os modelos adversariais condicionados (com base na eq.1.4) são usados quando o artista quer gerar imagens com as caraterísticas das imagens de treino, mas condicionadas por informação adicional, como outras imagens, faixas de áudio ou texto. Em 2017, Isola et al. introduzem um modelo adversarial condicionado para tradução imagem para imagem [24] e disponibilizam o código do modelo, designado por *pix2pix*. A ideia fundamental deste modelo é usar uma imagem adicional para condicionar a GAN a gerar imagens com caraterísticas específicas. Para fazer isso, os autores substituiram a variável c na eq.1.4 por uma imagem x. Consequentemente, o modelo é treinado com pares de imagens (x, w), que consistem na imagem de entrada x e a imagem de saída w. O gerador G recebe a imagem de entrada x e aprende a gerar uma imagem próxima de w, formando um par artificial$(x, G(x))$ que deve enganar o discriminador. O discriminador D é treinado para receber pares$(x, G(x))$ e (x, w) e avaliar se o par em causa é real ou artificial (fig.1.13).

Mario Klingemann[24] é um dos artistas que mais explora este modelo. Na maioria dos seus ensaios artísticos, usa o *pix2pix* treinado com pares semelhantes ao par representado na figura

[23] Os seus ensaios artísticos são publicados em http://annaridler.com/ e https://twitter.com/annaridler
[24] Os seus ensaios artísticos são publicados em https://twitter.com/quasimondo

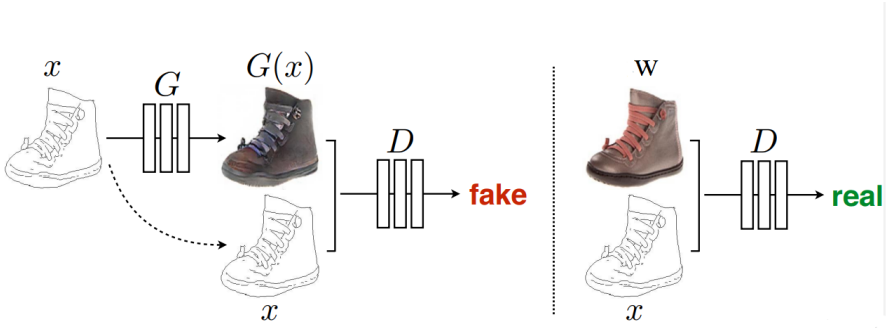


Figura 1.13: Esquema do modelo de tradução imagem para imagem. Fonte da imagem: [24]

1.14. Depois de obter a função geradora $G(x)$, avalia-a para um conjunto de imagens x (p.ex. os *frames* do vídeo da *webcam*), gerando a respetiva imagem ou vídeo no espaço representacional de G (fig.1.14 meio). Klingemann[25], nos últimos meses, focou-se no uso do *pix2pix* para tradução de imagens ou *frames* para retratos humanos, tendo conseguido resultados surpreendentes. Numa vertente mais minimalista, a artista Helena Sarin[26], treinou o *pix2pix* para tradução de fotografias complexas em desenhos simples, baseados em arestas e nuances de cor. Alguns dos seus resultados estão representados na (fig.1.14 baixo).

Existem mais dois modelos adversariais condicionados populares entre os artistas computacionais: os modelos de super-resolução (SR) e de previsão do próximo frame. Os modelos de super-resolução são treinados com pares (LR, HR) onde LR é a imagem de baixa resolução (imagem de entrada) e HR é a versão de alta-resolução da imagem LR (imagem de saída). O processo é idêntico ao do modelo de tradução imagem para imagem *pix2pix*. O gerador G aprende a mapear uma imagem de baixa-resolução na respectiva versão de alta resolução $G(LR)$, construindo um par artificial $(LR, G(LR))$ que deve enganar o discriminador. O discriminador recebe os pares (LR, HR) e $(LR, G(LR))$ e aprende a avaliar se o par em causa é real ou artificial. Exemplos de modelos SR usados na arte generativa são o LAPSRN [28] e o SRGAN [31]. Há dois tipos de uso destes modelos na arte computacional. O primeiro, e mais directo, é o aumento da resolução de uma dada imagem X, com um modelo SR treinado para aumentar a resolução de imagens da mesma classe que X. O segundo tipo, consiste no aumento da resolução de uma dada imagem X com um modelo SR treinado para aumentar a resolução de imagens de classes diferentes da classe de X. Vamos designar o primeiro tipo por *uso coerente* e o segundo por *uso incoerente*. Esta terminologia pode ser adotada para todos os modelos generativos condicionais, não sendo restrita aos modelos SR. Mike Tyka fez uso coerente de modelos SR para obter versões em alta resolução de retratos humanos gerados com DCGANs, criando um conjunto de retratos artificiais designado por *Neural Portraits*. Aqui o artista procura que o processo de mapear a imagem LR para HR seja o mais "limpo" possível, i.e. sem artefactos digitais (fig.1.15 lado esquerdo). O *uso coerente* de modelos SR serve para os artistas combaterem o problema da baixa resolução das imagens de

[25] https://twitter.com/quasimondo
[26] Os seus ensaios artísticos são publicados em: https://twitter.com/glagolista

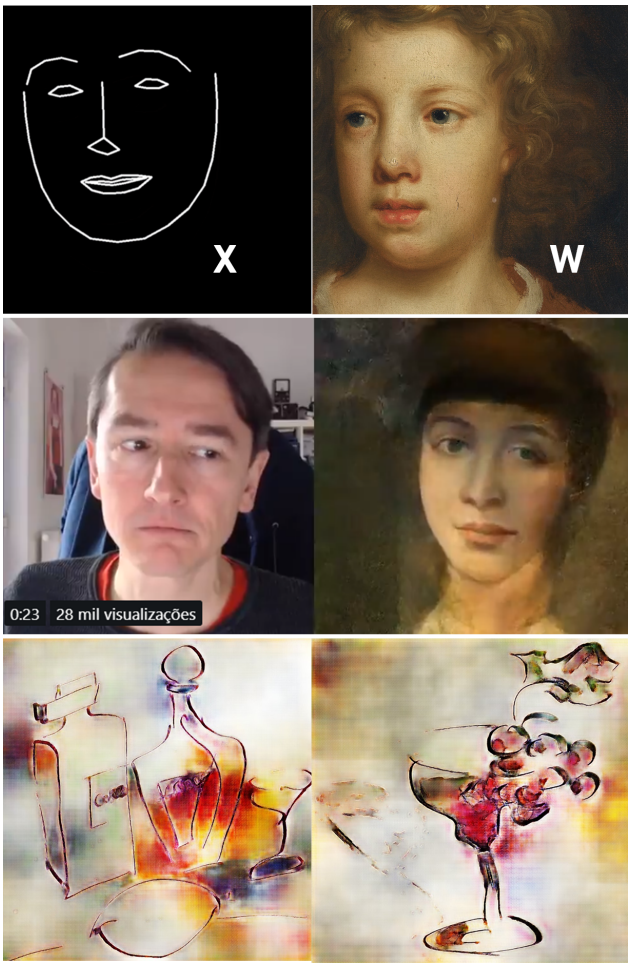


Figura 1.14: Cima: Um exemplo de uma par de treino (x, w). Meio: Mario Klingemann a usar os *frames* da *webcam* como imagens de entrada para o gerador treinado para gerar retratos a óleo. Baixo: Exemplos de obras de Helena Sarin geradas a partir de um modelo tradução imagem-imagem.

Figura 1.15: Lado esquerdo: Digital, *Neural Portraits*, 2017. Obra de Mike Tyka obtida através de uso coerente de um modelo de super-resolução. Lado direito: Sem título, 2018. Obra de Mario Klingemann obtida através de uso incoerente de um modelo de super-resolução.

saída dos modelos neurais generativos, por vezes limitados computacionalmente à resolução de 256×256 px. Mario Klingemann fez *uso incoerente* de modelos SR para aumentar a resolução de retratos a óleo de figuras humanas, com modelos treinados para aumentar a resolução de outro tipo de imagens. Este processo resulta, assim, no aparecimento de inúmeros artefactos digitais, deixando a versão HR da imagem original quase irreconhecível (fig. 1.15 lado direito). O objectivo do uso incoerente é o aparecimento de artefactos digitais que sejam esteticamente interessantes. O número de artefactos digitais será mais numeroso quanto maiores forem as diferenças entre as imagens de treino e as imagens de entrada usadas.

Nas últimas semanas, os modelos adversariais de previsão do próximo *frame* começaram a ser explorados mais intensamente pela comunidade artística. Estes modelos [32] são treinados com pares (f, nf) onde f é o *frame* de entrada e nf é o *frame* que procede o *frame* f, i.e. o *frame* de saída). O processo de treino é idêntico ao dos modelos anteriores. Os ensaios artísticos que têm surgido nesta vertente, são originados por um *uso incoerente* destes modelos. Mais concretamente, os modelos são treinados para prever *frames* de vídeos, por exemplo de carros a circular em estradas ou mergulhadores a filmar debaixo de água e, posteriormente, usados para prever os *frames* seguintes de imagens estáticas com naturezas completamente diferentes dos vídeos usados para treinar o modelo. Os vídeos produzidos resultam assim da evolução imprevisível de uma imagem estática , cuja dinâmica é construída a partir do *feedback loop* da função geradora. Mario Klingemann e Akmtn[27] são exemplos de artistas que têm vindo a explorar estes modelos.

1.3.5 Problemas dos modelos neurais na arte generativa computacional visual

Os modelos neurais discutidos permitem que os artistas computacionais usem redes neurais para um grande número de aplicações e obtenham resultados diversos. A construção de bibliotecas de treino e os diferentes usos (*coerentes* ou *incoerentes*) dados aos modelos, são os elementos

[27] Os seus ensaios artísticos são publicados em https://twitter.com/akmtn_twi

principais da criatividade do artista quando este os utiliza. No entanto, ainda existem problemas na aplicação dos modelos neurais que limitam as possibilidades dos artistas computacionais. Os problemas fundamentais, neste momento, são a previsibilidade e baixa resolução dos resultados e a limitação à geração de imagens *pixel-based*.

Baixa resolução dos resultados: A grande maioria dos modelos neurais estão limitados à geração de imagens com baixas resoluções (em geral, 256x256 px); isto deve-se à complexidade que esses modelos apresentam, não havendo *hardware* que permita a geração de imagens com resolução superior. Isto constitui uma grande limitação para os artistas, uma vez que não permite a geração de imagens com muitos pormenores nem a posterior impressão em grande escala. Como já referimos anteriormente Mike Tyka consegue combater este problema com o uso de modelos de super-resolução para aumentar as imagens geradas. No entanto, este método dá origem ao aparecimento de um certo número de artefactos digitais ou, no melhor dos casos, um aumento de resolução limpa mas sem um refinamento dos pormenores, como podemos ver no artigo [50]. No ano de 2017, foi proposto um modelo neural generativo designado por *Progressive GAN* [26] propõe uma solução para o problema da baixa resolução, no entanto exige *hardware* com grande computabilidade, que não está acessível para a maior parte dos artistas.

Previsibilidade dos resultados: Uma DCGAN, quando treinada com imagens de pinturas de paisagens, gera novas imagens de pinturas de paisagens com a estética intrínseca das primeiras. As novas paisagens podem apresentar composições inesperadas para o artista, no entanto a estética permanece; isto é claramente detectável nos trabalhos de Robbie Barrat e Anna Ridler. Isso pode ser a intenção do artista, no entanto, temos que admitir que seria igualmente interessante ter um sistema que, em vez de fazer uma reordenação percetual dos elementos das imagens de treino, se se inspirasse nessas imagens e criasse novas, tentando fugir da estética das imagens do treino. Uma DCGAN treinada com imagens aleatórias, leva a um treino instável capaz de produzir imagens com composições e estéticas aleatórias, no entanto, para este caso o artista não tem qualquer controlo na semântica final da imagem, não sendo um processo criativo interessante. O modelo CAN (Creative Adversarial Networks) [9], introduzido em 2017, é o modelo que combate mais eficazmente este problema. O discriminador usado neste modelo não só classifica a imagem de entrada como real/falsa, como a classifica quanto ao estilo. O gerador é treinado para gerar imagens que sejam classificadas como reais e que tenham igual probabilidade de pertencer a qualquer um dos estilos. Isto permite que o gerador gere imagens que tentam desviar-se dos estilos das imagens de treino, como por exemplo, estilo impressionista, cubista, etc. Para fazer isso, é introduzida uma função de perda adicional, designada *ambiguity loss*, que penaliza as imagens geradas que estejam mais vinculadas a um estilo c_k com gradientes mais altos, para que evoluam para imagens com um estilo diferente das imagens de treino. O gerador G procura assim minimizar a seguinte expressão,

$$\min_G \max_D - \sum_{k=1}^{K} \left(\frac{1}{K}\log(D_c(c_k|G(z))) + (1-\frac{1}{K})\log(1-D_c(c_k|G(z)))\right) \tag{1.6}$$

Figura 1.16: Exemplos de imagens geradas com a CAN. Fonte da imagem: [9]

que representa a entropia cruzada entre a distribuição uniforme e a distribuição $D_c(c_k|G(z))$. Esta entropia cruzada é mínima quando as classes c_k forem equiprováveis. A figura 1.16 contém algumas imagens geradas com este modelo para imagens de treino que englobam obras de arte desde o século XV até ao século XX.

Limitação à geração *pixel-based:* Todos os modelos que foram apresentados são construídos para gerar grelhas de píxeis. No entanto, a arte generativa computacional visual teve sempre duas abordagens de geração: geração *pixel-based* e geração *element-based*. A geração *element-based* consiste em gerar imagens através da organização de elementos no espaço, como círculos, linhas, triângulos, etc.; essa organização está sujeita a um conjunto de regras que são implementadas no computador. As obras de Casey Reas são exemplos clássicos desse tipo de geração. A construção de modelos neurais *element-based* permitiria integrar elementos no processo de treino, o que aumentava as possibilidades artísticas. Outra das vantagens seria o acesso ao espaço latente para imagens *element-based* o que permitiria a geração de animações coerentes desses elementos com distribuições orgânicas e complexas.

1.3.6 Questões acerca da aplicação dos modelos neurais na arte generativa visual

Face aos problemas anteriormente apresentados, duas questões surgem:

1. É possível criar um modelo neural que gere imagens que consistem em abstrações das imagens de treino, de modo a combater a previsibilidade dos resultados dos modelos neurais generativos usados actualmente?

2. É possível criar modelos neurais para geração de imagem baseada em elementos e que possam ser treinados com imagens *bitmap*?

No presente trabalho são introduzidos dois modelos neurais com o objetivo de explorar estes problemas e encontrar possíveis soluções. O primeiro modelo, designado por AGAN, permite a

geração de imagens com diferentes níveis de abstração das imagens de treino, quer na composição quer no estilo.

O segundo modelo, designado bezierGAN, permite a geração de imagens baseadas em linhas que são dispostas no espaço com base nas distribuições das imagens de treino. Este modelo é limitado à geração de imagens baseadas em linhas de bezier, porém apresenta uma *framework* a seguir para gerar imagens com outros elementos. O modelo é pesado computacionalmente, havendo ainda muitos problemas a serem investigados.

Capítulo 2

Construção de modelos neurais para a criação de arte generativa visual

2.1 AGAN: um modelo generativo de abstrações

2.1.1 Método

AGAN é um modelo que se inspira no processo cognitivo de abstração[1], visto na perspetiva de um processo de compressão [4], i.e na identificação de similiariedades entre objetos e ao processo de associar esses objetos numa só abstração, neste caso, numa só imagem. De modo a simular esse processo de abstração num modelo neural, utilizou-se uma arquitetura baseada em redes neurais adversariais. No entanto, em vez de ser utilizada a estrutura convencional gerador-discriminador, utilizou-se um discriminador binário adicional, designado abstrator. O discriminador é treinado com um conjunto de imagens X e o abstrator é treinado com um conjunto de imagens Y. Neste modelo o gerador é treinado para gerar imagens que tenham caraterísticas que simultaneamente ativem os filtros das camadas convolutivas dos dois discriminadores, chegando eventualmente a estados de equilíbrio. As caraterísticas de equilíbrio resultam da fusão das caraterísticas que são avaliadas em cada discriminador; consequentemente, as imagens geradas têm estilos e composições diferentes das imagens dos conjuntos X e Y consistindo em abstrações destas, na medida em que são uma representação das semelhanças profundas entre os dois conjuntos de imagens.

2.1.2 Arquitetura

A arquitetura deste modelo é baseada redes generativas adversariais (GANs) [15]. Como vimos anteriormente (sec.1.3.4), as GANs são compostas por dois modelos que são treinados para competir um com o outro. O gerador G é otimizado para reproduzir a distribuição dos dados de treino p_{data}, gerando imagens que são difíceis para o discriminador distinguir das imagens reais.

[1] Abstração é uma condição necessária para a cognição, uma vez que é a responsável por formar "imagens secundárias" da realidade, como perceções, conceitos e teorias. No processo de abstração, a escolha e processamento da informação é feita de modo a substituir a imagem empírica por outra, que não é direta, mas implícita e concebível como um objeto abstrato e geralmente chamada pelo mesmo termo "abstração".

O discriminador D é otimizado para distinguir as imagens reais das geradas por G. O processo como um todo, é governado pela equação,

$$\min_G \max_D V(D,G) = \min_G \max_D E_{x\sim p(x)}[\log D(x)] + E_{z\sim p(z)}[\log(1-D(G(z)))] \tag{2.1}$$

onde x é a imagem real com distribuição p_{data} e z é um vetor aleatório proveniente da distribuição p_z (geralmente uma distruibuição de *Gauss*).

AGAN usa esta arquitetura, no entanto com um discriminador A adicional, designado abstrator. Neste modelo o abstrator é otimizado para distinguir as imagens reais, com distribuição p_{dataY}, das imagens geradas por G. De maneira semelhante o discriminador é otimizado para distinguir as imagens reais, com distribuição p_{dataX}, das imagens geradas por G. A equação que descreve este processo é obtida a partir da eq.2.1 somando os termos relativos ao abstrator, i.e.

$$\min_G \max_D V(D,G) = \min_G \max_D E_{x_{dataX}\sim p(x)}[\log D(x)] + E_{y\sim p_{dataY}(y)}[\log A(y)]$$

$$+ \lambda_x E_{z\sim p(z)}[\log(1-D(G(z)))] + \lambda_y E_{z\sim p(z)}[\log(1-A(G(z)))] \tag{2.2}$$

onde x é uma amostra da distribuição p_{dataX} e y é uma amostra da distribuição p_{dataY}; λ_x e λ_y são constantes.

O discriminador adicional A faz com que o equilíbrio de Nash[2] da GAN [11] se desloque. Consequentemente, nesse ponto de equilíbrio o valor da divergência entre o distribuição do modelo e as distribuições p_{dataY} e p_{dataX} não é mínima, como acontece na GAN regular. De facto, a divergência é mínima entre a distribuição do modelo G e uma distribuição p' que depende de p_{dataY}, p_{dataX},λ_x e λ_y e resulta de um estado misto entre os dois discriminadores. Por este motivo, as imagens geradas pelo modelo têm estilos e composições diferentes das imagens iniciais de treino. O modelo "inspira-se" nas imagens de treino para criar abstrações destas. O esquema da arquitetura geral está representado na figura 2.1.

Arquitetura dos discriminadores A e D: Os modelos A e D consistem numa rede neural *feedfoward* com 4 camadas convolutivas e 1 *fully-connected layer* que faz a classificação final das imagens de entrada com volume $W\,(largura) \times H\,(altura) \times C\,(n^o canais)$. A camada convolutiva l tem $df * 2^{l-1}$ filtros, sendo que df é o número de filtros da primeira camada convolutiva ($l = 1$). Em cada camada convolutiva é feito um *downsampling* do volume de entrada num fator 2 seguido de *batch normalization* [9] e ativação *leakyReLU* como sugerido por Radford e Metz [40]. Este *downsampling* em cada camada é necessário para aumentar rapidamente o campo recetivo dos neurónios da próxima camada, de modo a que a camada $l = 4$ consiga avaliar caraterísticas mais globais das imagens de entrada. O *fully-connected layer* tem a função de ativação *sigmoid* e retorna um escalar com valores entre 0 e 1 (próximo de 0 se a imagem for classificada como falsa e próximo de 1 caso contrário). A arquitetura destes modelos está representada na figura 2.2.

[2] O equilíbrio de Nash representa, num jogo envolvendo dois ou mais jogadores, a situação na qual nenhum jogador tem a ganhar se mudar a sua estratégia unilateralmente.

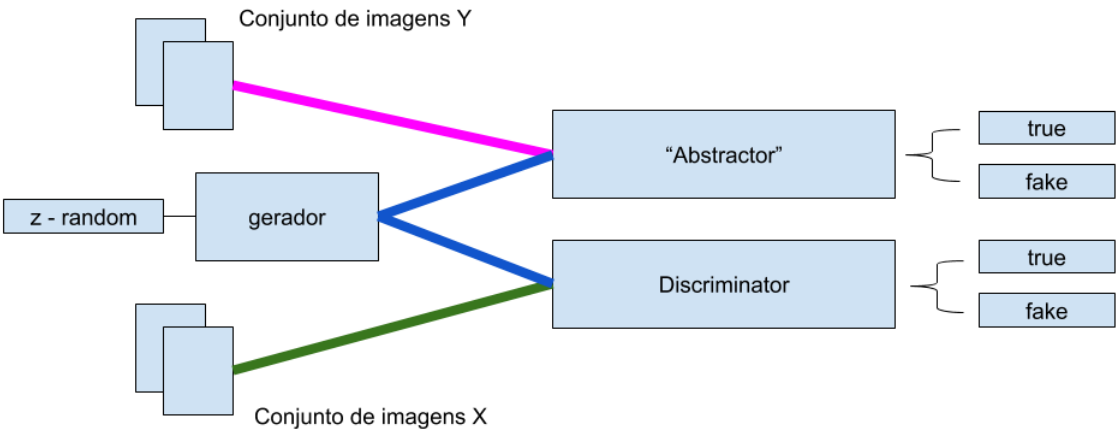


Figura 2.1: Esquema do modelo AGAN.

Arquitetura do gerador G: Para a construção do modelo G, também se utilizou as diretrizes sugeridas por Radford e Metz [40]. O modelo G consiste numa rede neural *feedfoward* com 1 *fully-connected layer* e 4 *transposed convolutional layers* (como representado na fig. 2.2). O modelo G recebe um vetor z (obtido através de uma distribuição de Gauss), que é conectado densamente, através do *fully-connected layer*, a um volume de $\frac{W}{16} \times \frac{H}{16} \times gf$, onde gf é o número de filtros da primeira camada convolutiva de G. Após isso, uma série de 4 *transposed convolutional layers* com $\frac{gf}{2^l}$ (para $l < 4$) filtros são aplicados, fazendo *upsamplings* sucessivos do volume de entrada, seguidos de *batch normalization* [23] e ativação ReLU. O último *transposed convolutional layer* ($l = 4$) tem apenas 3 filtros que correspondem aos 3 canais RGB da imagem de saída. A imagem de saída é normalizada por uma função de activação tanh. A arquitetura do modelo G está representada na fig.2.2.

2.1.3 Detalhes da implementação

As imagens de entrada de A e D e de saída de G têm resolução 128×128 ($W, H = 128$) e três canais RGB ($C = 3$). As camadas convolutivas dos modelos G, A e D têm filtros de dimensão 5×5 e *stride* igual a 2. O número de filtros da primeira camada do gerador é $gf = 512$ e para a primeira camada convolutiva de A e D é $df = 64$. O momento e o épsilon da *batch normalization* usados são iguais a 0.99 e 0.001 respetivamente. A otimização de todas as redes é feita com o *ADAM solver* com um *batch* de 16 imagens e *learning rate* de 0.0002. A implementação é feita com o uso da *framework Tensorflow*[3] e *cuDNN* v.7. A implementação do modelo está disponível em https://github.com/danielmachado93/AGAN.

[3]https://www.tensorflow.org/

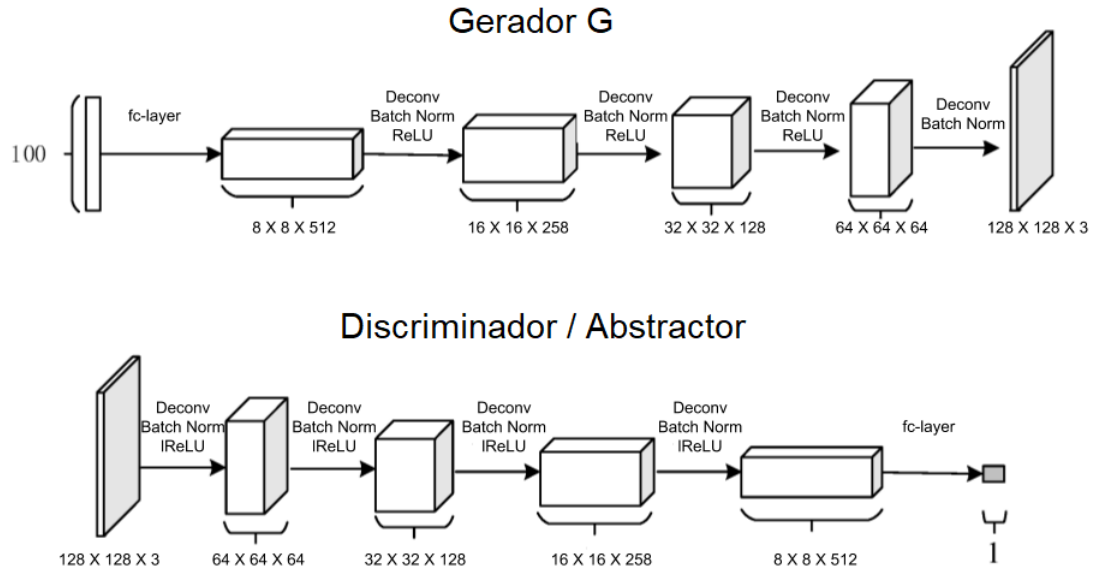


Figura 2.2: Arquitetura usada para os modelos G, D e A.

2.1.4 Dados de treino

A ideia principal deste modelo é a geração de abstrações das imagens de treino. No entanto, para um artista é importante controlar o objeto base dessa abstração, ou seja, ensinar o modelo a gerar abstrações de um objeto em específico. Com base nesta ideia, realizou-se um ensaio com um conjunto de imagens X, representativas de um objecto específico, para treinar o discriminador, e um conjunto de imagens Y, com imagens de categorias aleatórias para treinar o abstrator. Deste modo, o gerador é otimizado para gerar imagens que se aproximam simultânemente de imagens dum objecto específico e das imagens aleatórias. Realizou-se ainda um ensaio com dois conjuntos de imagens X e Y de objectos específicos para treinar o discriminador e o abstrator respetivamente. Com estes ensaios espera-se que as imagens geradas resultem da redução das caraterísticas das imagens presentes nos dois conjuntos X e Y, de modo a ficarem somente com as caraterísticas comuns entre as imagens, construindo assim uma abstração destas.

- Ensaio 1: o conjunto X consiste em 10,000 imagens de fotografias de faces de pessoas pertencentes à biblioteca *CelebA*; o conjunto Y consiste em 10,000 imagens de categorias aleatórias pertencentes à biblioteca Caltech 101 (fig. 2.3). Este experimento foi realizado para $\lambda_x, \lambda_y = 0.5$, para $\lambda_x = 0.1, \lambda_y = 0.9$ e $\lambda_x = 0.9, \lambda_y = 0.1$; os diferentes casos representam diferentes intensidades para o abstrator e o discrimador.

- Ensaio 2: o conjunto X consiste em 10,000 imagens de flores pertencentes à biblioteca *ImageNet*; o conjunto Y consiste em 10,000 imagens de pinturas de paisagem pertencentes à biblioteca *WikiArt* (fig. 2.5). Este experimento foi realizado para $\lambda_x, \lambda_y = 0.5$, para $\lambda_x =$

Figura 2.3: Exemplos de imagens dos conjuntos X e Y para os dois ensaios realizados.

$0.1, \lambda_y = 0.9$ e $\lambda_x = 0.9, \lambda_y = 0.1$; os diferentes casos representam diferentes intensidades para o abstrator e o discrimador.

2.1.5 Resultados finais

Os resultados obtidos para os dois ensaios, descritos anteriormente, encontram-se na figura 2.4 e figura 2.5. Uma imagem gerada pelo gerador G, para um vetor z fixo, é apresentada para cada *epoch*[4] de treino e para cada valor de $\lambda_x = \lambda_D$, $\lambda_y = \lambda_A$. São também apresentadas imagens geradas com os modelos AGAN, após serem treinados com os conjuntos de imagens de cada um dos ensaios para $\lambda_D = \lambda_A = 0.5$ (fig. 2.6).

2.1.6 Análise qualitativa do modelo

Os resultados obtidos, nos dois ensaios, demonstram que o modelo oferece controlo sobre o grau de abstração que queremos obter das imagens dos conjuntos X e Y. Para $\lambda_D = 0.9$, podemos constatar que a influência do abstrator é quase nula, uma vez que as imagens geradas são muito próximas das imagens geradas por uma DCGAN convencional quando o discriminador é treinado com o conjunto de imagens X. Neste caso, como $\lambda_A = 0.1$, o abstrator não força o gerador a capturar a distribuição das imagens do conjunto Y. Isto faz com que as imagens geradas tenham as distribuições dos conjuntos X, i.e. das imagens de faces de pessoas e das imagens de flores (fig. 2.4 e fig. 2.5 baixo).

[4]Passagem completa pelos dados de treino

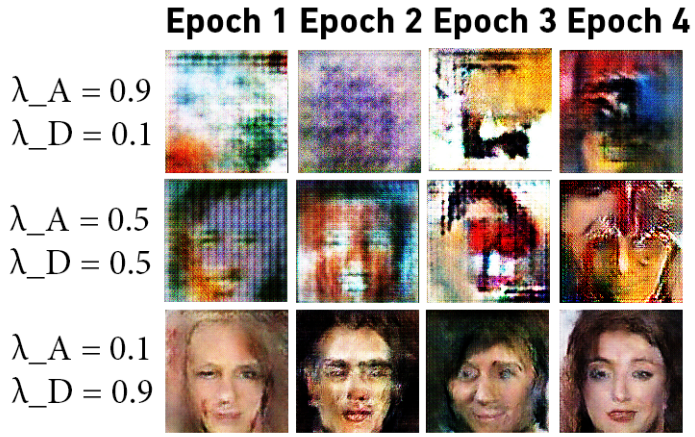


Figura 2.4: Resultados obtidos para o ensaio 1. Esta figura representa a imagem gerada pelo modelo, para o valor fixo de z, no final de cada *epoch* de treino.

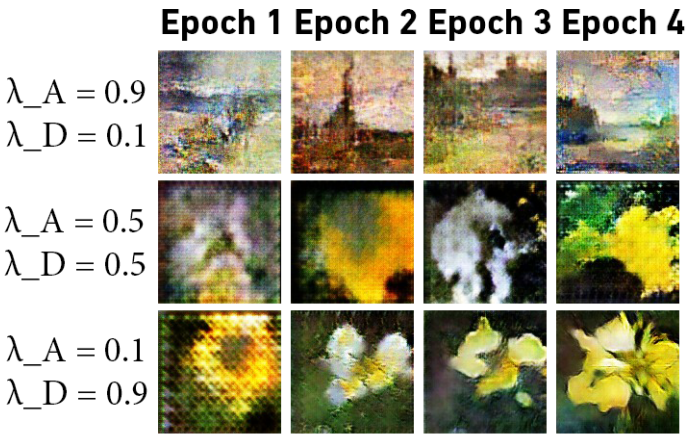


Figura 2.5: Resultados obtidos para o ensaio 1. Esta figura representa a imagem gerada pelo modelo, para o valor fixo de z, no final de cada *epoch* de treino.

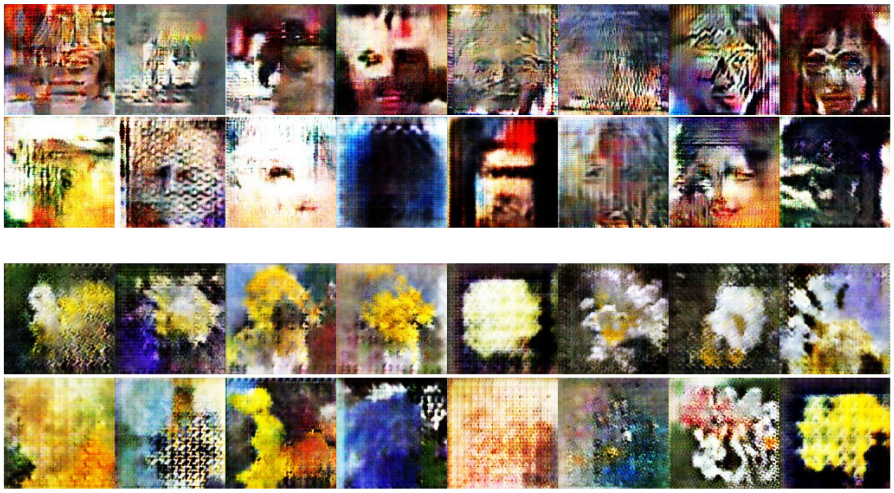


Figura 2.6: Imagens geradas com o modelo AGAN, após ser treinadas com os conjuntos X e Y de imagens de cada um dos ensaios para $\lambda_D = \lambda_A = 0.5$. (Ensaio 1 - cima, Ensaio 2 - baixo)

Para $\lambda_A = 0.9$, as imagens obtidas demonstram que o gerador é mais influenciado pelo abstrator do que pelo discriminador. Para este caso, as imagens geradas no ensaio 1 são pouco representacionais (fig.2.4). Isto é expectável, uma vez que, nesta situação, a AGAN é aproximadamente igual a uma DCGAN com um discriminador treinado com imagens de categorias aleatórias, e portanto, falha em capturar a distribuição real dos dados. Já para o ensaio 2, as imagens geradas mostram que o abstrator teve maior influência do que o discriminador, uma vez que representam pinturas de paisagem, que vão de encontro à distribuição das imagens do conjunto Y (fig.2.5 cima).

Para $\lambda_D = \lambda_A = 0.5$, as imagens produzidas demonstram que, em vez do sistema capturar a distribuição das imagens do conjunto X ou Y com mais predominância, captura uma distribuição que resulta da mistura de ambas. Como o conjunto X tem imagens de uma classe específica (faces humanas ou flores), o modelo consegue capturar informação mais coerente do discriminador do que do abstrator, conseguindo gerar "abstrações" das imagens do conjunto X. Essas "abstrações", como as figuras 2.4, 2.5 e 2.6 indicam, resultam da redução ou troca de caraterísticas das imagens dos conjuntos X e Y. No ensaio 1, as imagens geradas contêm as caraterísticas necessárias para reconhecer uma cara, como os olhos e a boca, todavia as caraterísticas menos fundamentais, como a cor, estilo e alguns elementos do rosto, são trocadas por outras caraterísticas que não estão presentes nas imagens do conjunto X. No ensaio 2, as imagens geradas têm caraterísticas e elementos menos identificáveis, uma vez que o conjunto Y, ao contrário do ensaio 1, é formado por imagens de uma classe específica (pinturas de paisagens). Consequetemente, o abstrator é otimizado eficazmente, o que faz com que o gerador capture uma distribuição igualmente próxima das distribuições das imagens dos conjuntos X e Y, gerando imagens mais abstratas.

O *blur* das imagens apresentadas deve-se ao reduzido tempo de treino, causado pela baixa computabilidade do hardware usado.

2.2 bezierGAN: modelo neural para arte generativa baseada em elementos

2.2.1 Método

Em 2017, David Ha apresentou uma rede neural recurrente *sketch-rnn* [17] capaz de gerar *sketches* de objetos (baseados em linhas) em formato vetorial e que é treinada com imagens em formato vetorial. No entanto, um modelo capaz de criar imagens vetorizadas baseadas noutros elementos e, acima de tudo, capaz de ser treinado com imagens *bitmap*, como fotografias, tem um maior potencial artístico e generativo, uma vez que a criação de bibliotecas de treino com milhares de imagens em formato vetorial é algo dificil de obter e trabalhar. O bezierGAN é um modelo experimental criado para explorar as possibilidades da criação de imagens baseadas em elementos vetorizados e que é treinado com imagens baseadas em píxeis. A implementação proposta do modelo permite a criação de imagens baseadas em curvas de Bézier cúbicas[5], porém pode ser generalizada para outro tipo de elementos, tais como retângulos, círculos, etc.

2.2.2 Arquitectura

O modelo bezierGAN consiste numa rede generativa adversarial [15], na qual o gerador G é treinado para gerar os pontos de controlo de N curvas cúbicas de bézier, ou seja, $4N$ pontos. À última camada do gerador G, é anexado um descodificador, pré-treinado para mapear os pontos de controlo de cada curva à sua respetiva representação no espaço dos píxeis. Este descodificador permite gerar as imagens *bitmap* representativas das N curvas de bézier que a função G retorna. Essas imagens são encaminhadas para o discriminador D, otimizado para distinguir as imagens reais das imagens geradas, de modo a prosseguir com o treino adversarial comum (como descrito na sec. 1.3.4). Depois do modelo estar treinado, a função $G(z)$ retorna conjuntos de N curvas de bézier ($4N$ pontos principais) que formam imagens vetorizadas baseadas nas distribuições das imagens de treino. Os pontos de controlo, que a função G retorna, podem ser usados para gerar essas imagens em qualquer resolução.

O modelo satisfaz a equação convencional da GAN; no entanto, como o gerador retorna pontos de controlo de curvas de bézier, é necessário um descodicador, que mapeie esses pontos para as respetivas curvas no espaço 2D de $W \times H$ píxeis (esse processo está ilustrado na fig. 2.7). A equação do modelo assume então a seguinte forma,

$$\min_G \max_D V(D,G) = \min_G \max_D E_{x\sim p(x)}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(dec(G(z))))] \tag{2.3}$$

onde dec é o descodificador.

[5] As curvas de Bézier são curvas paramétricas muito usadas na área da computação gráfica, para modelar curvas de forma suave através de pontos de controlo. No anexo F encontra-se a descrição completa destas curvas.

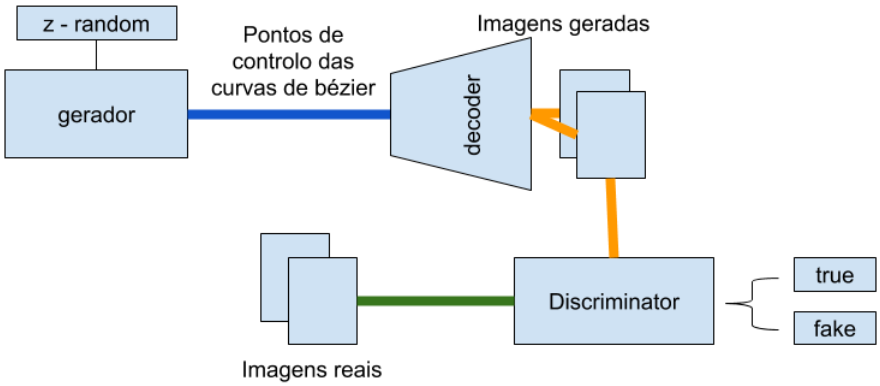


Figura 2.7: Esquema do modelo bezierGAN.

Arquitectura do discriminador D e gerador G As arquiteturas usadas para o discriminador D e o gerador G são exactamente iguais às arquiteturas usadas no modelo AGAN apresentadas na sec. 2.1.2. O gerador G, em vez de retornar grelhas 2D $W \times H$ correspondentes aos píxeis das imagens, retorna grelhas 2D $\sqrt{8N} \times \sqrt{8N}$ correspondentes às coordenadas (x,y) dos pontos principais das N curvas de bézier.

Arquitectura do descodificador A construção do descodificador é a verdadeira dificuldade na implementação do modelo. Como já foi referido, o objetivo do descodificador é mapear os 4 pontos de controlo de cada curva de bézier cúbica, na sua representação no espaço dos píxeis. Numa primeira abordagem, utilizou-se a matriz de *Bernstein*[6] para calcular as coordenadas por onde a curva passa, de modo a construir uma imagem a partir delas, dando o valor 1 aos píxeis subjacentes a essas coordenadas e 0 a todos os outros. No entanto, esta operação de atribuição é descontínua; isto faz com que os erros calculados a partir dessa imagem não possam ser retro-propagados através do descodificador e, seguidamente, para o gerador G e, portanto, a otimização do gerador não é possível. É necessário que o descodificador seja uma função contínua $dec(b)$ que mapeie um espaço ao outro. Como essa função é altamente complexa, usou-se uma rede neural para aproximá-la[7]. A arquitetura do descodificador está representada na figura 2.8. Para treinar o descodificador são necessários pares dos valores de entrada e de valores de saída; os valores de entrada são vectores b com as coordenadas cartesianas $(x_1,y_1,x_2,y_2,x_3,y_3,x_4,y_4) \in B$ dos 4 pontos de controlo da curva de bézier; os valores de saída são as imagens $y \in P_{W,H}$ que representam a curva de bézier no espaço dos píxeis com dimensão $W \times H$. Como o processo de otimização do descodificador é dificil, criou-se um codificador $enc(y)$, que mapeia o espaço $P_{W,H}$

[6] A matriz de Bernstein permite fazer a expansão da fórmula analítica das curvas de Bézier para a fórmula matricial. A descrição desta matriz está descrita no anexo F.

[7] teorema do aproximador universal

no espaço B, de modo a definir funções de perda mais eficazes para o treino do descodificador[8]. A arquitetura do codificador está representada na figura 2.8. O codificador é treinado com pares (y,b) : $y \in P_{W,H}$, $b \in B$ e tem como objectivo minimizar a seguinte função,

$$L_{enc} = \frac{1}{8}\sum_{i=1}^{8} |enc(y)_i - b_i| \tag{2.4}$$

que é simplesmente a média das distâncias L1[9] entre as coordenadas que deve retornar e as que efectivamente retorna.

O codificador, depois de otimizado, produz *feature maps* das imagens y que permitem-lhe deduzir as coordenadas dos 4 pontos de controlo da curva de bézier representada nessa imagem. Pode-se então usar os *feature maps* produzidos pelo codificador para definir uma função de perda eficaz para o decodificador. Seja enc^l os *feature maps* produzidos pelo bloco l[10] do codificador; o descodificador será otimo se os *features maps* $enc^l(\hat{y})$ das imagens que produz $\hat{y} = dec(b)$, forem idênticos aos *features maps* $enc^l(y)$ das imagens y que deve gerar, para um dado vector de coordenadas b. Isto é o equivalente a minimizar a média das distância L1 entre os *feature maps* da imagem gerada e da imagem de treino produzidos pelo codificador, i.e.,

$$L_{dec} = \frac{1}{W_l H_l}\sum_{i=1}^{W_l}\sum_{j=1}^{H_l} |enc^l(y)_{i,j} - enc^l(dec(b))_{i,j}| \tag{2.5}$$

Esta função de perda permite otimizar eficazmente o descodificador. Este tipo de função de perda, conhecida como *feature loss* ou *perceptual loss*, é utilizada em várias arquiteturas recentes [25, 21]. Depois de otimizado, o descodificador é utilizado para retornar as N imagens correspondentes aos N grupos de pontos de controlo produzidos pelo gerador. As imagens *bitmap* que representam cada curva são somadas de modo a gerar uma só imagem correspondente à sobreposição das N curvas de bézier (fig. 2.7). A imagem resultante representa toda a informação vetorizada gerada pelo gerador numa só imagem *bitmap*, que entra no discriminador e é comparada com as imagens *bitmap* reais, de modo a prosseguir com o treino adversarial.

[8] De um ponto de vista da teoria da informação, este decodificador descompressa sequências de 8 valores (correspondentes às 4 coordenadas dos pontos de controlo da curva) em sequências de $128 \times 128 = 16384$ valores (correspondentes à respectiva representação da curva no espaço $P_{128,128}$). Deste modo, esta descompressão pode ser altamente complexa, fazendo com que a rede neural nunca consiga convergir para a função que a representa. O processo inverso da descompressão é a compressão. Como as redes neurais convolutivas são ótimos classificadores, então o processo de compressão de uma imagem numa sequência de 8 valores é muito mais simples. Deste modo, podemos usar a informação proveniente do codificador para impor condições mais fortes para treinar o decodificador.

[9] Para um número real $p \geq 1$, a p-norma ou norma LP de x é definida como

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}.$$

A norma Euclidiana pertence a esta classe e é a distância L2 (ou 2-norma) e a distância L1 corresponde à distância retilínea.

[10] Este bloco é formado por três camadas que produzem volumes de saída com a mesma profundidade. O codificador tem 4 blocos, como podemos ver na fig. 2.8.

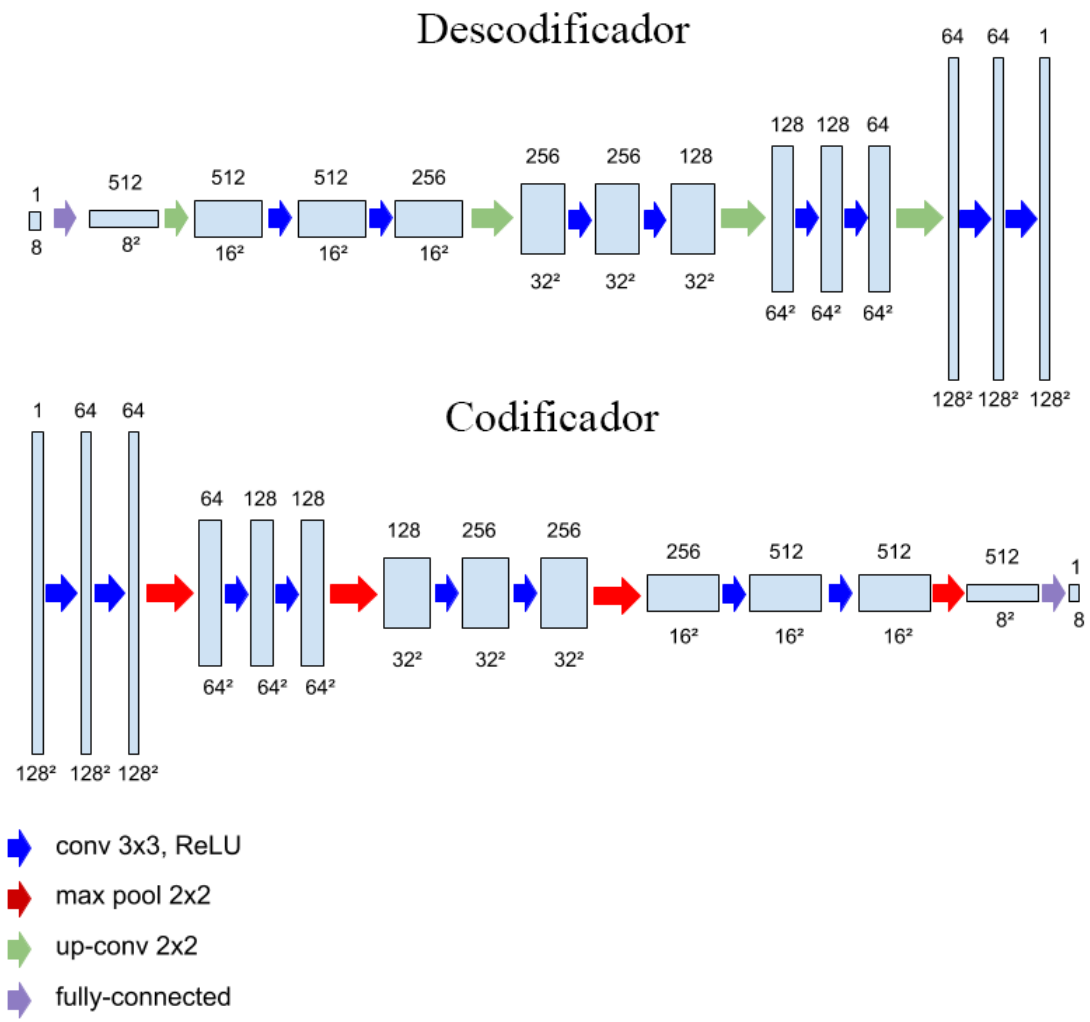


Figura 2.8: Arquitectura usada para o descodificador e codificador.

2.2.3 Detalhes da implementação

As camadas convolutivas dos modelos G e D têm filtros de dimensão 5×5 e *stride* igual a 2. O número de filtros da primeira camada do gerador é $gf = 512$ e do discriminador é $df = 64$. As camadas convolutivas do codificador e do descodificador têm filtros de dimensão 3×3 e *stride* igual a 1. O *downsampling* no codificador é feito com *maxpooling layers* com filtros 2x2 e *stride* 2. O *upsampling* no decodificador é feito com *transposed convolutional layers* com filtros 2x2 e *stride* 2. Os momentos e o épsilon da *batch normalization* usados são iguais a 0.99 e 0.001 respectivamente. A optimização das redes G e D é feita com o *ADAM solver* com um *batch* de 1 imagem e *learning rate* de 0.0002. A optimização das redes do codificador e decodificador é feita com o *ADAM solver* com um *batch* de 16 imagens/vetores, épsilon de 0.1 e *learning rate* de 0.001.

A implementação é feita com o uso da *framework Tensorflow*[11] e *cuDNN* v.7. A implementação do modelo está disponível em https://github.com/danielmachado93/bezierGAN.

2.2.4 Dados de treino

As imagens geradas são constituídas por curvas de bézier e nunca serão consideradas reais para um discriminador que é treinado com imagens complexas. Para contrariar isso, aplica-se nas imagens de treino um filtro passa-alto, de modo a preservar apenas as arestas das imagens. Deste modo as imagens de treino são transformadas numa versão mais simples, que pode ser obtida através de linhas. Só depois deste processo é que as imagens geradas e de treino estão em pé de igualdade para serem comparadas pelo discriminador, aumentando a probabilidade do treino adversarial ser bem sucedido.

O modelo foi treinado para dois conjuntos de imagens.

1. Ensaio 1: O primeiro conjunto consiste em imagens que representam um anel de pequena espessura (fig. 2.9 cima). Este experimento serve para testar se o modelo consegue localizar as 64 curvas de bézier de cada imagem dentro do anel.

2. Ensaio 2: O segundo conjunto consiste em 1000 imagens da biblioteca celebA (fotografias de faces humanas), sujeitas ao processo acima descrito (fig. 2.9 baixo). Este experimento permite testar se o modelo é capaz de dispor as curvas de bézier no espaço com distribuições mais complexas.

2.2.5 Resultados

Os resultados obtidos para os dois ensaios, descritos anteriormente, encontram-se na figura 2.10. Para cada ensaio são representados as imagens geradas pelo gerador ao longo do treino para dois valores de z fixos. Os intervalos entre as imagens é de 200 iterações e pertencem ao primeiro *epoch* de treino. Para cada imagem vetorial gerada é apresentada, no lado esquerdo (em fundo preto), a respetiva imagem *bitmap* proveniente do descodificador, necessária para calcular a perda do gerador relativamente ao discriminador.

[11] https://www.tensorflow.org/

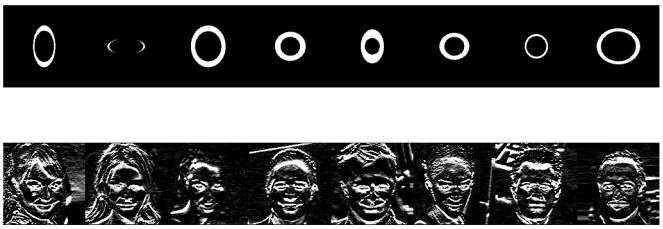


Figura 2.9: Exemplos das imagens de treino para o ensaio 1 (imagens de cima) e 2 (imagens de baixo).

2.2.6 Análise qualitativa do modelo

Os resultados obtidos demonstram que o modelo gera imagens baseadas em curvas de bézier dispostas com uma certa ordem, i.e. de maneira não aleatória, que depende da distribuição dos dados de treino, como podemos constatar através das diferenças das imagens representadas na figura 2.10 relativas ao ensaio 1 e 2, respectivamente. Por exemplo, as curvas de bézier das imagens geradas no ensaio 1 (fig. 2.10 lado esquerdo), são dispostas em formações circulares. Esse comportamento do modelo deve-se ao caráter dos anéis representados nas imagens de treino. Todavia, constata-se que essas imagens não são versões simplificadas, baseadas em curvas, das imagens de treino, uma vez que existem diferenças estruturais. Em suma, constata-se que a distribuição dos dados de treino influencia a ordem e orientação das curvas de bézier num certo grau, que é suficiente para dar "intenção" à imagem gerada, mas insuficiente para fazer uma representação fiel, embora simplicada, dos objetos das imagens de treino. As imagens obtidas com este modelo, contêm curvas de bézier que seguem o fluxo das imagens de treino e curvas mais predominantes que expressam caraterísticas mais presentes nas imagens de treino (p.ex. a fig. 2.10 lado direito e fig. 2.11 têm imagens onde podemos detetar alguns traços dos rostos humanos). O facto das imagens geradas não serem mais representativas das imagens de treino, i.e. com caraterísticas mais identificáveis, é justificado pelo reduzido número de curvas de bézier (igual a 64) utilizado nesta implementação. O número reduzido de curvas acessível ao gerador faz com que este gere imagens que nunca vão ser comparáveis às imagens reais de treino. Deste modo, o discriminador facilmente classifica estas imagens como falsas, fornecendo gradientes muito próximos de zero para o gerador, o que prejudica a otimização deste. Como esta implementação do modelo é pesada computacionalmente, o aumento do número de linhas é insustentável para os recursos computacionais que tinha disponíveis para a realização dos ensaios. No entanto, este comportamento do modelo é interessante, do ponto de vista artístico, uma vez que o modelo gera *sketches* que não são a cópia simplificada das imagens de treino. Deste modo, o modelo usa a distribuição das imagens de treino para dar uma "intenção" ao *sketch* que não é totalmente previsível.

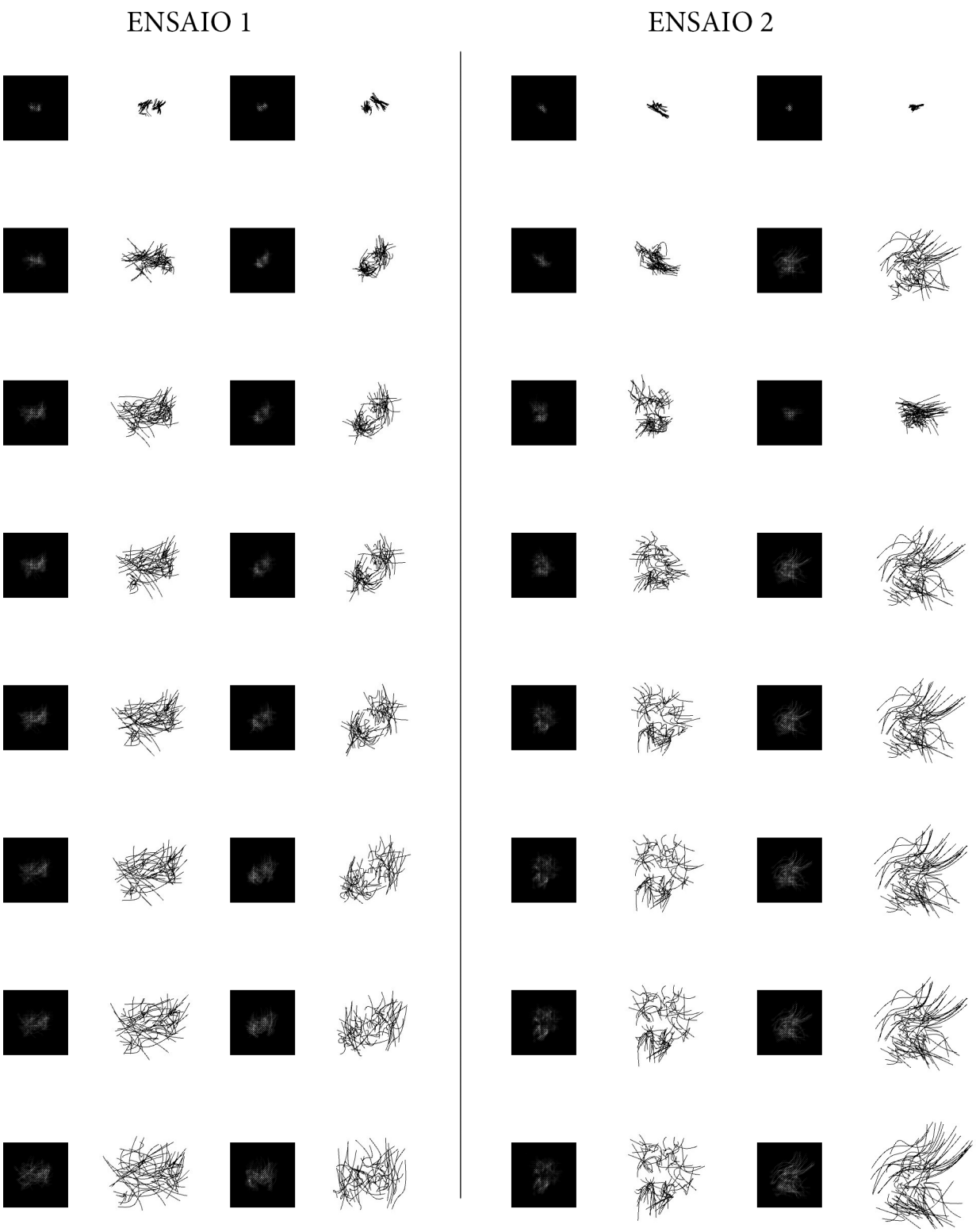


Figura 2.10: Resultados obtidos como o modelo bezierGAN para o ensaios 1 (lado esquerdo) e 2 (lado direito). As imagens em fundo preto representam as imagens resultantes da decodificação dos pontos principais produzidos pelo gerador nas respetivas imagens *bitmap*. As imagens em fundo branco representam a renderização das imagens vetorais geradas.

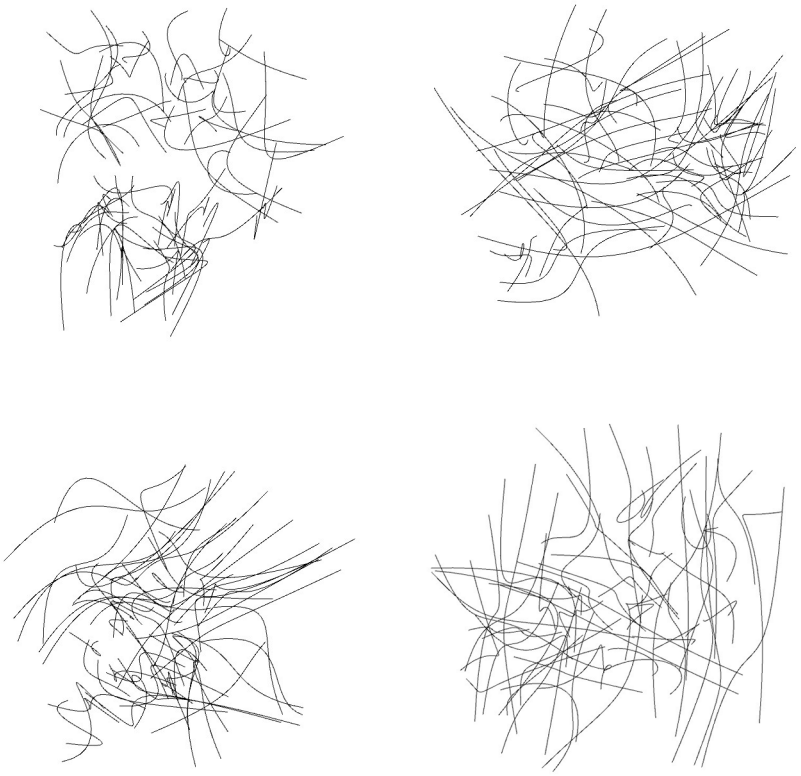


Figura 2.11: Renderização em alta resolução de algumas imagens vectoriais produzidas pela bezierGAN no ensaio 1 (lado esquerdo) e 2 (lado direito).

Capítulo 3

Conclusão e trabalho futuro

Neste trabalho foram descritos dois modelos generativos, designados AGAN e bezierGAN, que aumentam as possibilidades de aplicação das redes neurais na arte generativa visual. AGAN é um modelo baseado em DCGANs que permite criar abstrações das imagens de treino, através do uso de um discriminador adicional, designado abstrator. A existência de dois discriminadores, leva a que o modelo tenha que ser treinado com dois conjuntos de imagens distintos. Os resultados obtidos mostram que, se um dos discriminadores for treinado com um conjunto de imagens de uma classe específica de objetos, então o modelo é capaz de gerar imagens que representam "abstrações" desses objetos. Os resultados mostram ainda que o grau de abstração presente nas imagens geradas é controlável através dos parâmetros que definem a influência que cada discriminador têm sobre o gerador. O gerador é forçado a construir essas "abstrações", uma vez que gera imagens com caraterísticas presentes nos dois conjuntos de treino, de modo a minimizar, simultâneamente, a sua perda para cada um dos discriminadores. As imagens obtidas demonstram que o modelo não captura nenhuma das distribuições dos conjuntos, mas sim, uma distribuição que resulta da mistura de ambas. Deste modo, o modelo evita a previsibilidade dos resultados que gera, pois tenta encontrar "abstrações" das imagens de treino, em vez de lhes capturar a distribuição, ou seja, de tentar copiá-las. Por este motivo, o modelo é criativo, na medida em que cria imagens diferentes das imagens de treino, a partir das caraterísticas e semelhanças entre elas. O modelo CAN [9] partilha um objetivo próximo ao objetivo do modelo AGAN proposto nesta dissertação. No entanto, esse modelo usa uma função de perda designada *ambiguity loss* para penalizar as imagens geradas que pertençam a um determinado estilo de arte, forçando o gerador a produzir imagens que não pertençam, predominantemente, a nenhum dos estilos das imagens de treino. Consequentemente, o treino deste modelo é feito com etiquetas que indicam o estilo de arte a que cada imagem de treino pertence. Este tipo de treino é trabalhoso, sendo pouco versátil para os artistas. Pelo contrário, o modelo AGAN é mais versátil, uma vez que é treinado da mesma forma que os modelos já utilizados pelos artistas computacionais, i.e. apenas com imagens. A implementação deste modelo requer um elevado poder computacional, tal como acontece com as implementações de outros modelos baseados em redes neurais adversariais. Por este motivo, o problema da baixa resolução das imagens geradas existe também na aplicação do modelo AGAN. Os artistas podem contornar

Figura 3.1: Imagens geradas com o modelo AGAN, que foram aumentadas com um modelo de super-resolução pré-treinado e sujeitas a um pós-processamento. Esse pós-processamento, para a imagem do lado esquerdo, consiste numa simetrização, numa correção de cor e adição de *noise* e, para a imagem do lado direito, numa correção de cor e adição de *noise*.

este problema, com as técnicas que têm vindo a aplicar, i.e. aumentar a resolução das imagens com um rede neural de super-resolução e, posteriormente, fazer algum pós-processamento a estas, tal como adição de *sharp*, adição de ruído e correções de cor. Na figura 3.1 ilustro duas imagens geradas pelo modelo AGAN que passaram por este processo completo.

O modelo AGAN mostrou-se eficaz na geração de "abstrações" das imagens de treino, no entanto, as imagens obtidas, indicam que a implementação do modelo deve ser melhorada. As imagens geradas são, com frequência, desfocadas e quadriculadas. De modo a melhorar os resultados do modelo, irei, no futuro, explorar novas implementações deste. Essas implementações consistirão no aumento da profundidade do gerador e discriminadores e o uso das técnicas de otimização de GANs propostas por Wasserstein [1].

O modelo bezierGAN é um modelo que permite a geração de imagens vetoriais baseadas em curvas de bézier e que é treinado com imagens *bitmap*. A geração baseada em elementos é uma das práticas principais da arte generativa e este modelo incorpora-a dentro da arquitetura de uma DCGAN convencional. Os resultados obtidos mostram que o modelo gera imagens que consistem em curvas de bézier dispostas no espaço, com uma certa ordem, ditada pela distribuição dos dados de treino, mas que não é representativa dessas imagens. Desta forma, a distruibuição dos dados de treino não define o conteúdo das imagens de saída, mas apenas a ordem das curvas, que às vezes é marcada por algumas linhas principais presentes nas imagens de treino. Este fenómeno é causado pelo reduzido número de linhas acessível ao gerador, que não é suficiente para gerar uma imagem que seja comparável às imagens reais que passam pelo discriminador, causando o desaparecimento dos gradientes provenientes do discriminador necessários à otimização do gerador. Esse problema não pode ser combatido neste modelo, pois fica computacionalmente pesado para um número maior de linhas. Na prática, o modelo gera pequenos *sketches* de linhas, com um certo grau de organização (ou intenção) e dá acesso ao espaço latente das redes adversariais para imagens baseadas em elementos, tendo potencial para geração de vídeos e imagens generativas minimalistas em qualquer resolução. Este modelo é experimental e serve como ponto de partida e motivação para o desenvolvimento de modelos neurais generativos de imagens vetoriais

que são treinadas com imagens *bitmap*. No futuro irei explorar e desenvolver um modelo com o mesmo objetivo, que consistirá numa GAN condicionada e pré-treinada para mapear imagens *bitmap* nas respectivas vetorizações e numa rede neural adversarial para gerar vetorizações a partir das anteriores.

Bibliografia

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[2] Margaret A. Boden and Ernest A. Edmonds. What is generative art? *Digital Creativity*, 20(1-2):21–46, 2009.

[3] Cristian Bodnar. Text to image synthesis using generative adversarial networks. 2018.

[4] Gregory Chaitin. The limits of reason. *SCIENTIFIC AMERICAN, INC*, 2006.

[5] Yuhua Chen, Feng Shi, Anthony G. Christodoulou, Zhengwei Zhou, Yibin Xie, and Debiao Li. Efficient and accurate MRI super-resolution using a generative adversarial network and 3d multi-level densely connected network. *CoRR*, abs/1803.01417, 2018.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[7] Li Deng and Dong Yu. Deep learning: Methods and applications. Technical report, May 2014.

[8] Emily L. Denton, Soumith Chintala, Arthur Szlam, and Robert Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. *CoRR*, abs/1506.05751, 2015.

[9] Ahmed M. Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. CAN: creative adversarial networks, generating "art"by learning about styles and deviating from style norms. *CoRR*, abs/1706.07068, 2017.

[10] William Fedus, Ian Goodfellow, and Andrew Dai. Maskgan: Better text generation via filling in the . 2018.

[11] William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew Dai, Shakir Mohamed, and Ian Goodfellow. Many paths to equilibrium: Gans do not need to decrease a divergence at every step. 2018.

[12] Philip Galanter. What is generative art? complexity theory as a context for art theory. In *In GA2003 – 6th Generative Art Conference*, 2003.

[13] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[14] Abel Gonzalez-Garcia, Joost Weijer, and Yoshua Bengio. Image-to-image translation for cross-domain disentanglement, 05 2018.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[16] Eric Grinstein, Ngoc Q. K. Duong, Alexey Ozerov, and Patrick Pérez. Audio style transfer. *CoRR*, abs/1710.11385, 2017.

[17] David Ha and Douglas Eck. A neural representation of sketch drawings. *CoRR*, abs/1704.03477, 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[20] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[21] Xun Huang, Yixuan Li, Omid Poursaeed, John E. Hopcroft, and Serge J. Belongie. Stacked generative adversarial networks. *CoRR*, abs/1612.04357, 2016.

[22] D. Hubel and T. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[24] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[25] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[26] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[27] Martin Kemp. Latham's life-forms. *Nature*, 1998.

[28] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Fast and accurate image super-resolution with deep laplacian pyramid networks. *CoRR*, abs/1710.01992, 2017.

[29] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[30] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits.

[31] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[32] Xiaodan Liang, Lisa Lee, Wei Dai, and Eric P. Xing. Dual motion GAN for future-flow embedded video prediction. *CoRR*, abs/1708.00284, 2017.

[33] Jerry Liu, Fisher Yu, and Thomas A. Funkhouser. Interactive 3d modeling with a generative adversarial network. *CoRR*, abs/1706.05170, 2017.

[34] Penousal Machado and Amílcar Cardoso. Generation and evaluation of artworks.

[35] Phil Madgwick. Is ai really intelligent?, May 2017.

[36] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[37] Shravan Murali. An analysis on computer vision problems. *Medium*, 2017.

[38] Anh Nguyen, Jason Yosinski, Yoshua Bengio, Alexey Dosovitskiy, and Jeff Clune. Plug & play generative networks: Conditional iterative generation of images in latent space. *CoRR*, abs/1612.00005, 2016.

[39] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 12 2014.

[40] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[42] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[43] J. R. Searle. Minds, brains, and programs. *The Behavioral and Brain Sciences*, 3:417–457+, 1980.

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[45] Karl Sims. Artificial evolution for computer graphics. *SIGGRAPH Comput. Graph.*, 25(4):319–328, July 1991.

[46] Lee Spector and Adam Alpern. Criticism, culture, and the automatic generation of artworks. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*, AAAI'94, pages 3–8. AAAI Press, 1994.

[47] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[48] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[49] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[50] Mike Tyka. Superresolution with semantic guide, 2017.

[51] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016.

[52] Prateek Verma and Julius O. Smith. Neural style transfer for audio spectograms. *CoRR*, abs/1801.01589, 2018.

[53] Jinliang Zang, Le Wang, Zi-yi Liu, Qilin Zhang, Zhenxing Niu, Gang Hua, and Nanning Zheng. Attention-based temporal weighted convolutional neural network for action recognition. *CoRR*, abs/1803.07179, 2018.

[54] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[55] Hang Zhang, Kristin J. Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. *CoRR*, abs/1803.08904, 2018.

[56] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. *CoRR*, abs/1802.08797, 2018.

[57] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.

Anexo A - Componentes das CNN

Componentes das redes neurais convolutivas

Convolutional Layers (CONV-layers): Neste tipo de camadas cada neurónio está conectado a um conjunto local de neurónios da camada anterior. Este conjunto, conhecido como campo recetivo, tem dimensão igual à dimensão dos filtros (ou pesos partilhados) que constituem esta camada. Cada um destes filtros treináveis são, geralmente, pequenos espacialmente (largura e altura), mas extendem-se por toda a profundidade do volume de entrada. Tipicamente, o primeiro CONV-layer de uma rede neural convolutiva tem filtros com dimensão $5 \times 5 \times 3$ ou $3 \times 3 \times 3$, uma vez que o volume de entrada é uma imagem com profundidade de 3 canais RGB. A ativação de cada neurónio do CONV-layer é calculado através de um processo designado convolução, que consiste no produto escalar dos valores do campo recetivo desse neurónio com os valores do filtro, seguido da soma com uma constante designada *bias* (este processo está representado na figura 3.2). Ao "passar" os n filtros ao longo dos campos recetivos de todos os neurónios, são calculados n mapas de ativação conhecidos por *feature maps*. Consequentemente, a profundidade do volume de saída de um CONV-layer resulta do empilhamento dos *feature maps* tendo um valor igual a n. A largura e altura do volume de saída é igual à altura e largura dos *feature maps*, cuja dimensão depende dos hiperparâmetros *receptive field size*[1], *stride*[2] e *zero-padding*[3]. O volume de saída (V) de um CONV-layer pode então ser calculado através da dimensão do volume de entrada (I), do *receptive field size* (K), do *stride* (S) e *zero-padding* (P) aplicados, através da seguinte expressão:

$$V = \frac{I - K + 2P}{S + 1} \tag{3.1}$$

Durante o processo de retro-propagação dos erros, os CONV-layers, ajustam os parâmetros dos n filtros, de modo a construir filtros que, quando activados, resultam em informação útil para

[1] O hiperparâmetro *receptive field size* (ou *kernel size*), define a dimensão dos filtros, que é equivalente a definir as dimensões do campo receptivo. Obviamente que isto tem inflência na dimensão dos *feature maps*; quanto maiores forem os campos receptivos dos neurónios do CONV-layer, menor será o número necessário destes para preencher todo o volume de entrada, resultando em *feature maps* com pequenas dimensões.

[2] O hiperparâmetro *stride* controla o passo do "deslizamento" dos filtros pelas imagens de entrada, que é o equivalente a controlar o quão sobrepostos estão os campos receptivos. Quanto menor for o *stride*, mais sobrepostos são os campos receptivos do neurónios da camada. Nesse caso, é necessário um maior número neurónios do CONV-layer para ocupar todo o volume de entrada, resultando em *feature maps* com maior dimensão.

[3] O volume de entrada pode ser aumentado espacialmente por um certo número de dimensões de zeros definido pelo hiperparâmetro *zero-padding*. O aumento da dimensionalidade do volume de entrada, aumenta, directamente, a dimensionalidade dos *feature maps*.

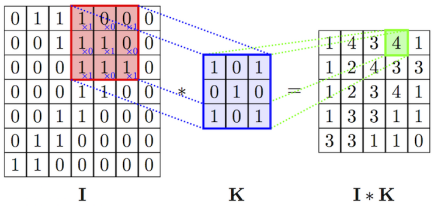


Figura 3.2: Activação de um CONV-layer.

a rede executar a classificação correta das imagens.

Fully-Connected Layers (FC-layers): Nestas camadas cada neurónio está conectado a todos os neurónios da camada anterior. Se o *FC-layer* tiver um número de neurónios C e camada de entrada tiver D ativações, então a ativação y_i do neurónio i do *FC-layer* l é calculada da seguinte forma:

$$y_i^{(l)} = f(z_i^{(l)}) \tag{3.2}$$

$$z_i^{(l)} = \sum_{k=0}^{m^{(l-1)}} w_{i,k}^{(l)} y_k^{(l-1)} + b_{i,0}^{(l)} \tag{3.3}$$

onde $w_{i,k}^{(l)}$ representa o peso da conexão (parâmetro treinável) entre o neurónio k da camda anterior $(l-1)$ e o neurónio i da camada (l) em causa. $b_{i,0}^{(l)}$ denota uma constante treinável externa somada a todas as conexões do neurónio i. $y_k^{(l-1)}$ representa o valor da ativação do neurónio k da camada anterior $(l-1)$. O parâmetro $m^{(l-1)} = D$ é o número de ativações da camada anterior $(l-1)$. Por fim, f denota a função de ativação usada no *FC-layer* l.

Note-se que esta fórmula só é válida no caso das ativações da camada anterior estiverem agrupadas num vetor unidimensional $y_k^{(l-1)}$. No caso da camada anterior ser, por exemplo, um *CONV-layer* esta fórmula já não é válida, uma vez que temos mapas 2D de ativação, i.e. *feature maps*. Nestes casos, faz-se tipicamente a conversão desses mapas 2D em vetores 1D de modo a usar a equação 3.3.

Pooling Layers (Poll-layers): A principal motivação dos *pooling layers* é a redução da dimensão espacial dos *features maps* da camada anterior. O exemplo mais básico é o *maxpooling layer*, no qual cada ativação resulta de selecionar a ativação máxima de um aglomerado de neurónios da camada anterior (fig. 3.3). Tipicamente, esse grupo local de neurónios tem uma dimensão 2x2 (*kernel size*) e são separados por uma passo igual a dois neurónios (*stride*); isto faz com que, para cada 4 ativações do *feature map* de entrada, tenhamos apenas uma ativação no *feature map* de saída, o que equivale a dizer que houve uma diminuição da resolução dos *feature maps* por um factor de 2.

Existem outros tipos de *pooling layers* que executam outras operações, como por exemplo, o *average pooling layer* que considera a média das ativações, em vez da ativação máxima.

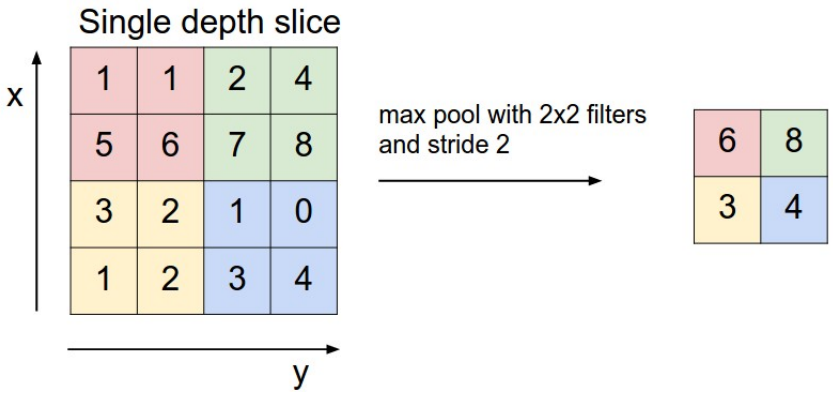


Figura 3.3: Operação de um max pooling layer.

Para um *pooling layer*, o volume de saída $W_o * H_o * D_o$ pode ser calculado através dos hiper-parâmetros *kernel size* (K) e *stride* (S) para um dado volume de entrada $W_i * H_i * D_i$ através das seguintes fórmulas:

$$W_0 = \frac{(W_i - K)}{S+1} \tag{3.4}$$

$$H_0 = \frac{(H_i - K)}{S+1} \tag{3.5}$$

$$D_o = D_i \tag{3.6}$$

Rectification Layers: Estas camadas resultam simplesmente de aplicar uma função de ativação a cada ativação presente nos *features maps* da camada anterior, i.e.,

$$y_i^{(l)} = f(y_i^{(l-1)}) \tag{3.7}$$

onde a função f é designada por retificador.

O retificador mais usado, é a função $f(x) = x^+ = max(0,x)$, dando origem a *features maps* com valores positivos ou iguais a zero, quebrando a linearidade do sistema, o que reduz a probabilidade de *overfitting*. As camadas que usam esse retificador são designados por ReLU-layers. Esta função de ativação, ao contrário da função *sigmoid* ou tangente hiperbólica, não tem o problema dos gradientes ficarem reduzidos a zero para a maior parte dos parâmetros treinavéis, acelerando a velocidade do treino. Por outro lado, esta função também é computacionalmente mais barata do que as restantes, acelerando a computação.

Anexo B - Arquitecturas CNN

Arquitecturas CNN mais usadas para extração de características

VGG network

Esta arquitetura foi introduzida por Simonyan e Zisserman em 2014 [44] e foi a primeira a lidar com bibliotecas de imagens de larga escala como a ImageNet [6]. As *VGG-nets* consistem em grupos de camadas convolucionais com um número crescente de filtros 3×3 ao longo da rede. A redução do volume é feito por *maxpooling layer*s. No final existem três *fully-connected layers*; os primeiros dois têm 4,096 nodos[4] e o último tem 1000 nodos, que são activados por uma função *softmax*. As 1000 activações resultantes da camada de saída representam as 1000 categorias possíveis para a classificação. A arquitetura da *VGG-19* está esquematizada na figura 3.4. A tabela 3.1 representa as diferentes composições possíveis, para 11, 16 e 19 camadas treináveis.

Residual network

Em 2015, Kaiming He introduz a Residual Network, que usa *identity mappings* e *batch normalization*[5] [23]. Os *identity mappings* (fig. 3.5), estabelecem uma ligação entre as duas extremidades de um bloco de camadas, ao passar os *features maps* de entrada para a última camada. Esses *features maps* são somados elemento-a-elemento ao volume de saída da último camada do bloco. Isto nasce da hipótese que múltiplas camadas de neurónios artificiais podem aproximar assintoticamente qualquer função não linear[6]. Se $H(x)$ for a função original que esse bloco de camadas tem que aproximar assintoticamente, então é possível que se aproxime assintoticamente da função residual $F(x) = H(x) - x$. Uma vez estabelecido o *identity mapping*, podemos calcular o volume de saída da última camada do bloco da seguinte forma:

$$y = F(x, \{w_i\}) + x \tag{3.8}$$

Embora o volume de saída bloco seja numericamente igual ao original $H(x)$, esta micro-arquitectura permite que o gradiente seja retro-propagado eficazmente através da váriavel x, i.e. do *identity mapping*. Isto resolve o problema do desaparecimento dos gradientes para arquitecturas

[4] Nome alternativo para os neurónios artificiais que constituem um rede neural.

[5] Layer de normalização, introduzido por Sergey Ioffe, que centra todas as activações relativamente à média, reduzindo a covariância dos valores; isto permite a redução do overfitting e o uso de maiores *learning rates*.

[6] Teorema do aproximador universal.

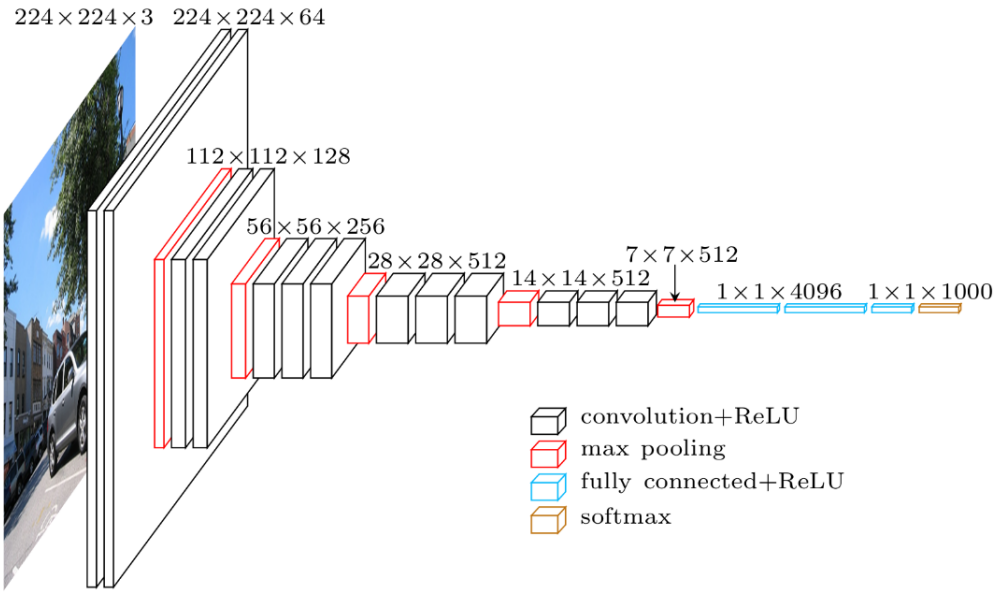


Figura 3.4: Esquema da arquitectura da VGG-19.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Tabela 3.1: Tipos de composições das *VGG-nets* apresentadas por Simonyan e Zisserman [44].

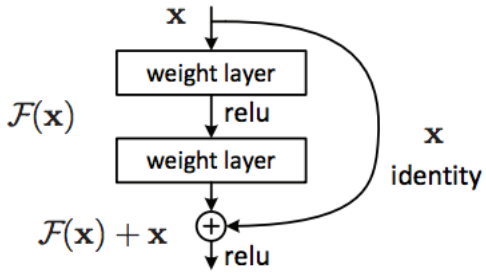


Figura 3.5: *Identity mapping* entre as extremidades de um bloco de layers.

profundas e, ao mesmo tempo, preserva a informação ao longo da rede. A Residual Network (ResNet) com 34 camadas, organizadas em blocos residuais, tem uma complexidade comparável a VGG-16 (número de parâmetros aproxidamente igual), porém apresenta um erro consideralvelmente menor [18].

Em 2016, Kaiming He, faz um estudo mais pormenorizado dos blocos residuais e propõe um bloco residual mais eficaz [19]. Em 2017, Gao Huang propõe as DenseNets[20] onde leva a ideia dos *identity mappings* ao extremo, fazendo conexões densas entre todas as pilhas de camadas, conseguindo os resultados de estado-de-arte actuais.

Inception networks

As arquiteturas Inception vN usam o módulo *Inception* originalmente introduzido por Christian Szegedy [48] em 2014 e usado pioneiramente na GoogLeNet (ou *Inception V1*) de 22 camadas. O objetivo do módulo *Inception* (fig 3.6) é fazer uma extração simultânea dos *features maps* com diferentes níveis extraídos com filtros de 1x1, 3x3 e 5x5. Os *features maps* resultantes das diferentes camadas convolutivas são concatenados, de modo a criar um só volume de saída e, seguidamente, enviados para outro módulo *Inception*. Esta arquitetura permite ter um número significativo de camadas convolutivas, sem aumentar descontroladamente a complexidade do sistema. Em 2015, Szegedy propõe uma versão melhorada da GoogLeNet, designada *Inception V3* e apresenta uma reformulação do módulo *Inception*[49].

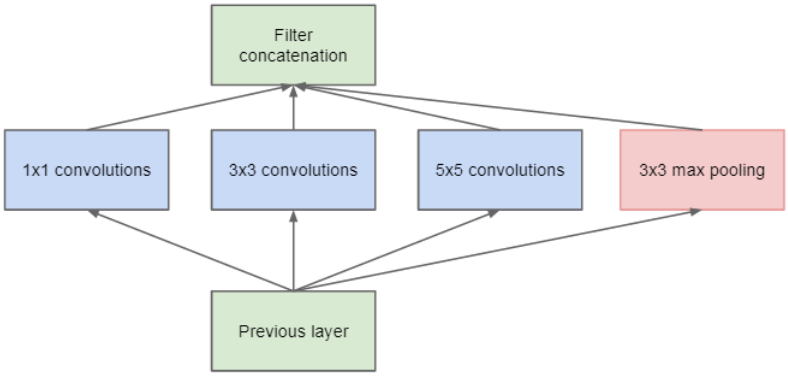


Figura 3.6: Módulo *Inception* - versão naíve.

Anexo C - Algoritmo do *DeepDream*

Algoritmo do *DeepDream*

O algoritmo do *DeepDream* usa um ciclo de *feedback* de modo a transformar uma imagem progressivamente até que as caraterísticas que um CNN identifica nela estejam vísiveis. Primeiramente, a imagem é avaliada numa GoogLeNet[7] e são selecionados os *feature maps* resultantes de um módulo Inception[8], por exemplo o terceiro (como representado na fig. 3.7). Esses *features maps*, representam as ativações de certas caraterísticas da imagem original. De modo a visualizar as caraterísticas que a CNN identifica na imagem, é necessário amplificá-las no espaço do píxeis. Para isso consideramos que os *feature maps* são os gradientes (∂y_{ij}) que devem ser retropropagados até ao espaço dos píxeis (imagem de entrada x), mantendo fixos todos os parâmetros da rede. Deste modo, visualizamos o efeito das activações presentes nos *features maps* na imagem de entrada. Os gradientes $dx = \frac{\partial x}{\partial y_{ij}} \partial y_{ij}$ obtidos são, seguidamente, multiplicados por uma constante λ, que controla a amplificação do efeito, e somados à imagem inicial.

$$\bar{x} = x + \lambda dx \tag{3.9}$$

Este processo é então repetido com a imagem $\bar{x}$ obtida.

[7] descrita no Anexo B.

[8] Este módulo é descrito no anexo B. Ele permite a extração de características em vários níveis representacionais, através de filtros de 1×1, 3×3 e 5×5.

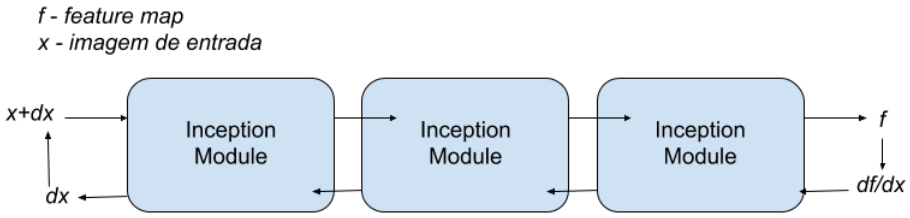


Figura 3.7: Esquema do algoritmo do DeepDreams.

Anexo D - Algoritmo de transferência de estilo

Algoritmo de transferência de estilo proposto por Gatys

Representação do Conteúdo: Consideremos que a imagem x é a imagem a ser gerada com o conteúdo da imagem designada por p. Consideremos ainda, que F^l,P^l são os repectivos *features maps* da camada l resultantes da codificação dessas imagens numa rede neural convolutiva como, por exemplo, a *Inception V1*. Se a imagem x deve apresentar o conteúdo da imagem p, então os *feature maps* F^l e P^l das camadas mais profundas têm que ser assintoticamente próximos, uma vez que são estes que constituem a representação do conteúdo. O função de perda entre os *feature maps* das duas imagens é,

$$L_{content}(p,x,l) = \frac{1}{2}\sum_{i,j}(F^l_{i,j} - P^l_{i,j}) \tag{3.10}$$

A derivada desta função de perda em relação aos *feature maps* da imagem x, é

$$\frac{\partial L_{content}(p,x,l)}{\partial F^l_{i,j}} = \begin{cases} (F^l_{i,j} - P^l_{i,j}) & ,F^l_{i,j} > 0 \\ 0 & ,F^l_{i,j} < 0 \end{cases} \tag{3.11}$$

Este gradiente é retro-propagado através das camadas de uma CNN para determinar o gradiente de $L_{content}$ em ordem a x (espaço dos píxeis) . O gradiente $\frac{\partial L_{content}}{\partial x}$ resultante é usado iterativamente para transformar a imagem x inicial, até que as camadas l produzam, para esta imagem, os mesmos *feature maps* que produz para a imagem p. Isto é equivalente a dizer que o conteúdo da imagem p está a ser transferido (através das derivadas) para a imagem gerada x.

Representação do estilo: Para obter a representação do estilo, Gatys usa um *feature space* que captura apenas a informação da textura da imagem, descartando qualquer informação relativa à organização espacial desta. Esse *feature space* é construído através das correlações entre as ativações presentes nos *features maps* $F^l_{i,j}$. Essas correlações são obtivas através das matrizes de Gram:

$$G^l_{i,j} = \sum_k F^l_{i,k} F^l_{j,k} \tag{3.12}$$

O conjunto destes *style feature spaces* $G^l_{i,j}$ captura a informação relativa ao estilo da imagem. Consideremos, então, que a imagem x é a imagem a ser gerada com o estilo da imagem designada por a e que $G^l_{i,j}$ e $A^l_{i,j}$ são os *style feature spaces* dessas imagens, respectivamente. Novamente, se a imagem x deve apresentar o estilo da imagem a, então os *style feature spaces* $G^l_{i,j}$ e $A^l_{i,j}$ têm que ser assintoticamente próximos. O função de perda entre os *styles feature spaces* das duas imagens é,

$$L_{style}(a,x,l) = \sum_l w_l E_l \tag{3.13}$$

onde,

$$E_l = \sum_{i,j} (G^l_{i,j} - A^l_{i,j}) \tag{3.14}$$

Posteriormente é calculada a derivada $\frac{\partial L_{style}(p,x,l)}{\partial F^l_{i,j}}$ que é retro-propagada ao longo da CNN até ao espaço dos píxeis. O gradiente $\frac{\partial L_{style}(p,x,l)}{\partial x}$, resultante da retro-propagação, é somado à imagem inicial x, de modo a minimizar a função de perda entre os *style feature spaces*, o que equivale a dizer que a imagem inicial x é transformada numa imagem com o estilo da imagem a.

Combinação das duas representações: O algoritmo final usa uma função de perda, que combina as duas representações, i.e.,

$$L_{total} = \alpha L_{content} + \beta L_{style} \tag{3.15}$$

A derivada $\frac{\partial L_{total}(p,x,l)}{\partial x}$ (calculada como nos casos anteriores) é somada à imagem x.

Através da repetição deste processo, a imagem x, que é inicialmente uma imagem com valores RGB valores aleatórios, converge sucessivamente para uma imagem que tem o estilo da imagem a e o conteúdo da imagem p. O processo inteiro está esquematizado na figura 1.10. Os parâmetros α e β controlam a percentagem de estilo e conteúdo que é transferido das imagens a e p para a imagem x.

Anexo E - Redes neurais adversariais

Redes Neurais Adversariais

A rede neural generativa consiste num modelo generativo G que captura a distribuição dos dados e um modelo discriminativo D que estima a probabilidade de uma amostra ser proveniente dos dados de treino ou do modelo G, i.e., de ser gerada artificialmente. Os modelos G e D são, usualmente, *multi-layer perceptrons*.

Para aprender a distribuição p_g dos dados de treino x, o gerador constrói uma função que mapeia uma distribuição de *noise* $p_z(z)$ para o espaço dos dados x, i.e, $G(z;\theta_g)$. O discriminador $D(x;\theta_d)$, retorna um escalar que representa a probabilidade de x ser proveniente dos dados de treino em vez do gerador G.

O discriminador D é treinado de modo a maximizar a probabilidade de retornar valores próximos de 1 para as amostras reais e próximos de 0 para as amostras geradas por G. Como a função $D(x)$ é contínua, os valores não podem ser apenas 0 ou 1, havendo valores intermédios. Consequentemente é necessária uma maximização da entropia cruzada entre a distribuições $p(x)$ e $D(x)$ e entre as distribuições $p(z)$ e $1-D(G(z))$, que é o equivalente a fazer com que a probabilidade do discriminador retornar os valores correctos (i.e. mais próximos de 1 e mais próximos de 0) seja máxima para as amostras mais frequentes. Valores como 0.5, que são inconclusivos são atribuídos às amostras menos frequentes. Durante o treino do discriminador, os parâmetros θ_d são ajustados de modo a minimizar a seguinte função:

$$\max_D V(D,G) = \max_D E_{x\sim p(x)}[\log D(x)] + E_{x\sim p(z)}[\log(1-D(G(z)))] \tag{3.16}$$

Por outro lado, o gerador G pretende gerar amostras G(z) que "enganem" o discriminador D. Para tal é necessário que os parâmetros θ_g sejam ajustados de modo a minimizar o termo $E_{x\sim p(z)}[\log(1-D(G(z)))]$, o que equivale a forçar o gerador G a aprender a gerar amostras com a distribuição das amostras reais x, de modo a que a probabilidade do discriminador atribuir valores próximos de 1 é maior. O processo inteiro é descrito pela seguinte equação,

$$\min_G \max_D V(D,G) = \min_G \max_D E_{x\sim p(x)}[\log D(x)] + E_{x\sim p(z)}[\log(1-D(G(z)))] \tag{3.17}$$

que é equivalente à equação de evolução de um jogo minimax entre dois jogadores.

As redes neurais adversariais podem ser extendidas a um modelo condicional [36] se o gerador G e o discriminador D forem condicionados por uma informação extra y, tal como uma etiqueta ou dados de outras modalidades. Esta extensão permite condicionar o gerador com informação extra y, de modo a controlar o tipo de amostras que gera. A informação y é concatenada ao valor de entrada do gerador e às imagens de entrada do discriminador. O treino adversarial, neste caso tem como objectivo a seguinte equação,

$$\min_G \max_D V(D,G) = \min_G \max_D E_{x\sim p(x)}[\log D(x|y)] + E_{x\sim p(z)}[\log(1-D(G(z|y)))] \tag{3.18}$$

Anexo F - Curvas de Bézier

Curvas de Bézier

As curvas de Bézier são curvas polinomiais paramétricas expressas através da interpolação linear entre um certo número de pontos designados por pontos de controlo. As curvas de Bézier resultam da combinação linear de polinómios de Berstein.

A base dos polinómios de Bernstein de ordem n é definida como

$$b_{v,n}(x) = \binom{n}{v} x^v (1-x)^{n-v}, \quad v = 0, \ldots, n$$

onde $\binom{n}{v}$ é o coeficiente binomial.

A base de polinómios de Bernstein de ordem n forma uma base do espaço vetorial Π_n dos polinómios de grau n.

A combinação linear de bases de polinómios de Bernstein

$$B_n(x) = \sum_{v=0}^{n} \beta_v b_{v,n}(x)$$

é chamada de polinómio de Bernstein ou polinómio na forma de Bernstein de grau n. Os coeficientes β_v são designados por coeficientes de Bernstein ou coeficientes de Bézier.

As curvas de Bézier de ordem n são polinómios de Bernstein de ordem n quando os coeficientes β_v representam os pontos de controlo da curva $\vec{P}_v$. Isto é,

$$Bezier_n(t) = \sum_{v=0}^{n} \vec{P}_v b_{v,n}(t)$$

Curvas de Bézier lineares (n=1):

Neste caso a curva de Bézier representa a linha reta entre os dois pontos de controlo $\mathbf{P}_0$ e $\mathbf{P}_1$. A curva é dada por

$$\mathbf{Bezier}_1(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \, , \, t \in [0,1]$$

e é equivalente à interpolação linear entre dois pontos.

Curvas de Bézier quadráticas (n=2):

A curva de Bézier quadrática representa a trajetória traçada pela função $\mathbf{Bezier}_2(t)$, dados três pontos de controlo $\mathbf{P}_0$, $\mathbf{P}_1$ e $\mathbf{P}_2$,

$$\mathbf{Bezier}_2(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2 \ , t \in [0,1].$$

Curvas de Bézier cúbicas (n=3):

A curva de Bézier cúbica representa a trajetória traçada pela função $\mathbf{Bezier}_3(t)$, dados três pontos de controlo $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ e $\mathbf{P}_3$,

$$\mathbf{Bezier}_3(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3 \ , t \in [0,1].$$

Esta é a classe de curvas de Bézier usada no modelo bezierGAN apresentado nesta dissertação.

Formulação matricial das curvas de Bézier cúbicas

A curva de Bézier cúbica pode ser escrita na forma matricial através da expansão da definição analítica. Isto é,

$$\begin{aligned}
\mathbf{Bezier}_3(t) &= (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3 \\
&= \begin{bmatrix} (1-t)^3 & 3t(1-t^2) & 3t^2(1-t) & t^3 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} \\
&= \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix}
\end{aligned}$$

A curva de Bézier cúbica pode ser então escrita da seguinte foma,

$$\mathbf{Bezier}_3(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} M \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix}$$

onde M é a matriz de Bernstein de ordem 3.